

# Cincom

## **AD/ADVANTAGE**

MANTIS Language  
OS/390, VSE/ESA

P39-5002-00



---

## **AD/Advantage®**


### **MANTIS Language OS/390, VSE/ESA**

**Publication Number P39-5002-00**

© 1992–1998, 2001 Cincom Systems, Inc.  
All rights reserved

This document contains unpublished, confidential, and proprietary information of Cincom. No disclosure or use of any portion of the contents of these materials may be made without the express written consent of Cincom.

The following are trademarks, registered trademarks, or service marks of Cincom Systems, Inc.:

AD/Advantage®	iD CinDoc™	MANTIS®
C+A-RE™	iD CinDoc Web™	Socrates®
CINCOM®	iD Consulting™	Socrates® XML
Cincom Encompass®	iD Correspondence™	SPECTRA™
Cincom Smalltalk™	iD Correspondence Express™	SUPRA®
Cincom SupportWeb®	iD Environment™	SUPRA® Server
CINCOM SYSTEMS®	iD Solutions™	Visual Smalltalk®
	intelligent Document Solutions™	VisualWorks®
gOOi™	Intermax™	

All other trademarks are trademarks or registered trademarks of:

Acucobol, Inc.	Micro Focus, Inc.
AT&T	Microsoft Corporation
Compaq Computer Corporation	Systems Center, Inc.
Data General Corporation	TechGnosis International, Inc.
Gupta Technologies, Inc.	The Open Group
International Business Machines Corporation	UNIX System Laboratories, Inc.
JSB Computer Systems Ltd.	

or of their respective companies.

Cincom Systems, Inc.  
55 Merchant Street  
Cincinnati, OH 45246-3732  
U. S. A.

PHONE: (513) 612-2300  
FAX: (513) 612-2000  
WORLD WIDE WEB: <http://www.cincom.com>

---

#### **Attention:**

Some Cincom products, programs, or services referred to in this publication may not be available in all countries in which Cincom does business. Additionally, some Cincom products, programs, or services may not be available for all operating systems or all product releases. Contact your Cincom representative to be certain the items are available to you.

---

---

## Release information for this manual

*AD/Advantage MANTIS Language, OS/390, VSE/ESA, P39-5002-00*, is dated October 30, 2001. This document supports Release 5.5.01 of MANTIS.

### We welcome your comments

We encourage critiques concerning the technical content and organization of this manual. At your convenience, please take the [survey](#) provided with the online documentation.

### Cincom Technical Support for AD/Advantage

*All customers*

Web: <http://supportweb.cincom.com>

*U. S. A. customers*

Phone: 1-800-727-3525

FAX: (513) 612-2000

Attn: AD/Advantage Support

Mail:

Cincom Systems, Inc.  
Attn: AD/Advantage Support  
55 Merchant Street  
Cincinnati, OH 45246-3732  
U. S. A.

*Customers outside U. S. A.*

All: Visit the support links at  
<http://www.cincom.com> to find  
contact information for your nearest  
Customer Service Center.



# Contents

<b>About this book</b>	<b>xiii</b>
Using this document .....	xiii
Document organization .....	xiii
Revisions to this manual .....	xiv
Conventions .....	xv
MANTIS documentation series .....	xviii
Educational material .....	xix
 <b>Overview of MANTIS language</b>	 <b>21</b>
Text considerations .....	22
Symbolic names .....	24
Numeric considerations .....	27
DBCS considerations .....	29
Signing on to MANTIS .....	29
 <b>MANTIS conventions</b>	 <b>31</b>
Programming fundamentals .....	31
Automatic mapping .....	33
Statements .....	36
Commands .....	38
Comments .....	39
Numeric data .....	41
Numeric literals and variables .....	41
Scientific notation (E-notation) .....	42
Arithmetic arrays .....	44
Arithmetic expressions .....	45
Text data .....	50
Text literals and variables .....	50
Text expressions .....	55
Relational text expressions .....	58
DBCS considerations .....	62
Built-in functions .....	63

<b>MANTIS programming language</b>	<b>75</b>
MANTIS language summary .....	76
ABS .....	86
ACCESS .....	87
ASI .....	93
ATN .....	94
ATTRIBUTE .....	95
ATTRIBUTE (Function) .....	95
ATTRIBUTE (Statement) .....	102
BIG .....	134
BREAK .....	136
CALL .....	137
CHAIN .....	139
CHR .....	144
CLEAR .....	145
COMMIT .....	149
COMPONENT .....	153
CONVERSE .....	157
General considerations .....	160
COS .....	164
CSIOPTNS .....	165
CURSOR .....	171
DATAFREE .....	176
DATE (Function) .....	177
DATE (Statement) .....	179
DBCS (Statement)(Kanji users only) .....	181
DELETE .....	183
DELETE (External file) .....	183
DELETE (MANTIS file) .....	189
DELETE (Personal computer file) .....	194
DELETE (RDM logical view) .....	197
DELETE (TOTAL file view) .....	200
DEQUEUE .....	203
DO .....	206
DOLEVEL .....	209
E .....	210
ENQUEUE .....	211
ENTRY-EXIT .....	213
EXEC_SQL-END .....	217
EXIT (Command) .....	219
EXP .....	220
FALSE .....	221
FILE .....	222
FOR-END .....	226
FORMAT .....	230
FSI .....	232

GET .....	234
GET (External file) .....	234
MANTIS external VSAM KSDS nonunique alternate key processing .....	244
GET (MANTIS file) .....	249
GET (Personal computer file) .....	254
GET (RDM logical view) .....	259
GET (TOTAL file view) .....	265
HEAD .....	270
HELP .....	272
IF-ELSE-END .....	274
INSERT .....	277
INSERT (External file) .....	277
INSERT (MANTIS file) .....	280
INSERT (Personal computer file) .....	283
INSERT (RDM logical view) .....	287
INSERT (TOTAL file view) .....	291
INT .....	294
INTERFACE .....	295
KANJI (Kanji users only) .....	298
KEY .....	301
KILL .....	303
LANGUAGE (Function) .....	305
LANGUAGE (Statement) .....	306
LET (Numeric (BIG/SMALL) variables) .....	308
LET (TEXT/KANJI/DBCS variables) .....	312
LOG .....	320
LOWERCASE .....	321
LUID .....	323
MARK (SUPRA RDM users only) .....	324
MIXD .....	327
MIXM .....	328
MIXMODE .....	329
MIXT .....	331
MODIFIED .....	332
NEXT .....	335
NOT .....	336
NULL .....	338
NUMERIC .....	339
OBTAIN .....	341
ORD .....	343
OUTPUT .....	344
PAD .....	346
PASSWORD .....	349

PERFORM .....	350
PERFORM transfers control to another program without passing program variables .....	353
PERFORM transfers control to an external program without a return .....	356
PERFORM transfers control to another program and saves MANTIS context without a return .....	358
PERFORM starts a MANTIS program as a background task .....	361
PERFORM starts a non-MANTIS program as a background task .....	366
PI .....	367
POINT .....	368
PRINTER (Function) .....	370
PRINTER (Statement) .....	371
PROGFREE .....	372
PROGRAM .....	373
PROMPT .....	376
RELEASE (Function) .....	378
RELEASE (Statement) .....	380
REPLACE .....	383
RESET .....	387
RETURN .....	388
RND .....	389
RUN .....	391
SCREEN .....	393
SCROLL .....	396
SEED .....	398
SGN .....	399
SHOW .....	400
SIN .....	403
SIZE .....	404
SLICE .....	410
SLOT .....	413
SMALL .....	415
SOURCE .....	417
SQLCA (Function) .....	420
SQLCA (Statement) .....	424
SQLDA (Function) .....	426
Read header elements .....	426
Read repeating elements .....	429
SQLDA (Statement) .....	432
Allocate an SQLDA .....	432
Deallocate an SQLDA .....	435
Set header information .....	438
Set repeating element information .....	444
SQR .....	451
STOP .....	452
TAN .....	454



TERMINAL .....	455
TERMSIZE .....	456
TEXT .....	457
TIME (Function).....	460
TIME (Statement) .....	462
TOTAL .....	464
TOTAL (TOTAL and SUPRA PDM users only) .....	464
TRAP .....	469
TRUE .....	472
TXT .....	473
UNPAD .....	474
UNTIL-END .....	478
UPDATE .....	479
UPDATE (External file).....	479
UPDATE (MANTIS file) .....	483
UPDATE (Personal computer file).....	486
UPDATE (RDM logical view) .....	488
UPDATE (TOTAL file view) .....	491
UPPERCASE.....	493
USAGE .....	495
USER.....	497
USERWORDS.....	498
VALUE .....	499
VIEW .....	501
VSI .....	505
WAIT .....	506
WHEN-END.....	508
WHILE-END .....	510
ZERO.....	512
 <b>Dissimilarity debugging</b>	 <b>513</b>
 <b>MANTIS reserved words</b>	 <b>515</b>

<b>Status functions</b>	<b>517</b>
RDM status functions .....	517
Function Status Indicators .....	518
Attribute Status Indicators .....	519
Validity Status Indicators .....	520
Extended status messages for MANTIS and external files.....	521
File status codes and messages.....	522
CICS MANTIS FSI message text descriptions for internal and external files or views .....	524
MANTIS for batch FSI message text descriptions for internal and external files or views .....	525
PC CONTACT FSI message text descriptions for internal and external files .....	526
 <b>Advanced programming techniques</b>	 <b>527</b>
External DO.....	527
Using external DO.....	530
Parameter passing.....	531
Program architecture .....	531
Internal DO vs. external DO vs. CHAIN .....	533
External DO programming guidelines .....	538
VSAM deadlocks.....	550
VSAM Files .....	550
Deadlocks on GET NEXT .....	551
Rules for avoiding deadlocks .....	553
 <b>Enhanced screen and program design</b>	 <b>555</b>
Designing screens.....	556
Building a map set in your program .....	561
The CONVERSE statement and mapping examples .....	563
Multiple images of a single screen design .....	567
Windowing .....	569
Window mode .....	571
Clearing a map.....	574
Clearing a map set.....	575
Advanced editing.....	578

<b>Mixed-data support</b>	<b>579</b>
Using mixed-data in your program .....	580
Using mixed-data in screen design .....	581
Heading fields .....	582
Screen design output and input and SO/SI pairs .....	583
Mixed-data expressions .....	585
MIXMODE statement .....	585
Literals and variables .....	585
Concatenation .....	586
Deconcatenation .....	587
Subscripts .....	588
Literals and mixed-data expressions .....	589
Built-in functions .....	590
SIZE .....	590
POINT .....	591
MIXM .....	591
MIXD .....	592
MIXT .....	592
Statements and commands .....	593
LET .....	593
SHOW .....	594
ATTRIBUTE .....	595
PAD .....	595
UNPAD .....	596
 <b>Glossary of terms</b>	 <b>597</b>
 <b>Index</b>	 <b>615</b>



# About this book

---

## Using this document

MANTIS is an application development system that consists of design facilities (e.g., screens and files) and a programming language. This manual describes the commands, functions, and statements of the language as well as some code examples.

### Document organization

The information in this manual is organized as follows:

#### **Chapter 1—Overview of MANTIS language**

Provides an overview of MANTIS programming language and the content of this manual.

#### **Chapter 2—MANTIS conventions**

Discusses MANTIS language conventions.

#### **Chapter 3—MANTIS programming language**

Describes the MANTIS programming statements, commands, and functions in alphabetical order, including the CEF statements.

#### **Appendix A—Dissimilarity debugging**

Defines the types of dissimilarity and suggests how you can locate and correct these errors.

#### **Appendix B—MANTIS reserved words**

Contains lists of reserved words as they were added with each new release of MANTIS.

### **Appendix C—Status functions**

Describes the meanings of several types of statuses that are returned by the status indicators FSI (Function Status Indicators), ASI (Attribute Status Indicators), and VSI (Validity Status Indicators).

### **Appendix D—Advanced programming techniques**

Presents advanced programming techniques that combine more than one statement.

### **Appendix E—Enhanced screen and program design**

Describes the MANTIS Screen Design Facility, which enables you to design and save screens for use in your application.

### **Appendix F—Mixed-data support**

Provides information on mixed-data—data streams composed of any combination of SBCS (Single-Byte Character Set), such as EBCDIC and DBCS (Double-Byte Character Set).

### **Glossary of terms**

### **Index**

## **Revisions to this manual**

For release 5.5.01, Cincom added the LUID built-in function. For more information on this function, see the [table](#) under “**Built-in functions**” (the table starts on page 63).

## Conventions

The following table describes the conventions used in this document series:

Convention	Description	Example
Constant width type	Represents screen images and segments of code.	Screen Design Facility GET NAME LAST INSERT ADDRESS
Yellow-highlighted, red code or screen text	Indicates an emphasized section of code or portion of a screen.	00010 ENTRY COMPOUND 00020 .SHOW"WHAT IS THE CAPITAL AMOUNT?" 00030 .OBTAIN INVESTMENT 00040 EXIT
Slashed b (b)	Indicates a space (blank).  The example indicates that a password can have a trailing blank.	WRITEPASSb
Brackets [ ]	Indicate optional selection of parameters. (Do not attempt to enter brackets or to stack parameters.) Brackets indicate one of the following situations.  A single item enclosed by brackets indicates that the item is optional and can be omitted.  The example indicates that you can optionally enter a program name.	COMPOSE [program-name]
	Stacked items enclosed by brackets represent optional alternatives, one of which can be selected.  The example indicates that you can optionally enter NEXT, PRIOR, FIRST, or LAST. (NEXT is underlined to indicate that it is the default.)	<div> <div>NEXT</div> <div>PRIOR</div> <div>FIRST</div> <div>LAST</div> </div>

Convention	Description	Example
Braces { }	<p>Indicate selection of parameters. (Do not attempt to enter braces or to stack parameters.) Braces surrounding stacked items represent alternatives, one of which you must select.</p> <p>The example indicates that you must enter FIRST, LAST, or a value for <i>begin</i>.</p>	<div><div>FIRST</div><div><i>begin</i></div><div>LAST</div></div>
<u>Underlining</u> (In syntax)	<p>Indicates the default value supplied when you omit a parameter.</p> <p>The example indicates that if you do not specify ON, OFF, or a row and column destination, the system defaults to ON.</p>	<div>SCROLL<div><div>ON</div><div>OFF</div><div>[<i>row</i>][<i>col</i>]</div></div></div>
	<p>Underlining also indicates an allowable abbreviation or the shortest truncation allowed.</p> <p>The example indicates that you can enter either PRO or PROTECTED.</p>	<div>PROTECTED</div>
Ellipsis points...	<p>Indicate that the preceding item can be repeated.</p> <p>The example indicates that you can enter (A), (A,B), (A,B,C), or some other argument in the same pattern.</p>	<div>(<i>argument</i>,...)</div>



Convention	Description	Example
UPPERCASE	<p>Indicates MANTIS reserved words. You must enter them exactly as they appear.</p> <p>The example indicates that you must enter CONVERSE exactly as it appears.</p>	CONVERSE <i>name</i>
<i>Italics</i>	<p>Indicate variables you replace with a value, a column name, a file name, and so on.</p> <p>The example indicates that you can supply a name for the program.</p>	COMPOSE [ <i>program-name</i> ]
Punctuation marks	<p>Indicate required syntax that you must code exactly as presented.</p> <p>( ) parentheses  . period  , comma  : colon  ; semicolon  ' single quotation mark  " " double quotation marks</p>	$[\text{LET}]_r \begin{bmatrix} (i) \\ (i, j) \end{bmatrix} [\text{ROUNDED}(n)] = e1 [, e2, e3 \dots]$

## MANTIS documentation series

MANTIS is an application development system designed to increase productivity in all areas of application development, from initial design through production and maintenance. MANTIS is part of AD/Advantage, which offers additional tools for application development. Listed below are the manuals offered with MANTIS in the IBM® mainframe environment, organized by task. You may not have all the manuals listed here.

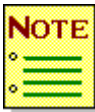
### MASTER User tasks

- ◆ *MANTIS Installation, Startup, and Configuration, MVS/ESA, OS/390, P39-5018*
- ◆ *MANTIS Installation, Startup, and Configuration, VSE/ESA, P39-5019*
- ◆ *MANTIS Administration, OS/390, VSE/ESA, P39-5005*
- ◆ *MANTIS Messages and Codes, OS/390, VSE/ESA, P39-5004\**
- ◆ *MANTIS Administration Tutorial, OS/390, VSE/ESA, P39-5027*
- ◆ *MANTIS XREF Administration, OS/390, VSE/ESA, P39-0012*

### General use

- ◆ *MANTIS Quick Reference, OS/390, VSE/ESA, P39-5003*
- ◆ *MANTIS Facilities, OS/390, VSE/ESA, P39-5001*
- ◆ *MANTIS Language, OS/390, VSE/ESA, P39-5002*
- ◆ *MANTIS Program Design and Editing, OS/390, VSE/ESA, P39-5013*
- ◆ *MANTIS Messages and Codes, OS/390, VSE/ESA, P39-5004\**
- ◆ *AD/Advantage Programming, P39-7001*
- ◆ *MANTIS DB2 Programming, OS/390, VSE/ESA, P39-5028*

- ◆ *MANTIS SUPRA SQL Programming, OS/390, VSE/ESA*, P39-3105
- ◆ *MANTIS XREF, OS/390, VSE/ESA, OpenVMS*, P39-0011
- ◆ *MANTIS Entity Transformers*, P39-0013
- ◆ *MANTIS DL/I Programming, OS/390, VSE/ESA*, P39-5008
- ◆ *MANTIS SAP Facility, OS/390, VSE/ESA*, P39-7000
- ◆ *MANTIS WebSphere MQ Programming*, P39-1365
- ◆ *MANTIS Application Development Tutorial, OS/390, VSE/ESA*, P39-5026



---

Manuals marked with an asterisk (\*) are listed twice because you use them for both MASTER User and general tasks.

---

## Educational material

AD/Advantage and MANTIS educational material is available from your regional Cincom education department.



# 1

## Overview of MANTIS language

MANTIS is a comprehensive application development system designed to increase productivity in all areas of application development—from initial design through production. MANTIS offers design facilities, prototyping capabilities, testing and debugging tools, and an advanced, high-level programming language. All MANTIS facilities are completely interactive. This means that once a program, screen, or file, for example, is created it is immediately available for display and review by end users. This eliminates the need for precompiling, compiling, binding, coding Job Control Language and other activities normally associated with application development.

This manual discusses the MANTIS programming language that you use to write your applications. This is not a manual about how to program. Instead, this manual provides the basics of the MANTIS programming language for you to apply to your current level of programming knowledge. The manual is divided into three chapters. “[Overview of MANTIS language](#)” on page 21 introduces the basic considerations of the MANTIS language such as how it handles text, numeric, and DBCS data. “[MANTIS conventions](#)” on page 31 takes these topics, describes them in more detail, and introduces the built-in functions of MANTIS. “[MANTIS programming language](#)” on page 75, representing the bulk of the manual, presents each command, function, and statement (in alphabetical order) with its syntax, format, and any special considerations. This alphabetical listing is the main reference point for most users. (For information on using the Program Design Facility and the editing commands you can use to modify your programs, refer to [MANTIS Program Design and Editing, OS/390, VSE/ESA, P39-5013](#).)

# Text considerations

The MANTIS character set consists of:

- ◆ Alphabetic characters A–Z
- ◆ Space character
- ◆ Numeric digits 0–9
- ◆ Special characters, as listed in the following table

The following table describes MANTIS special characters and the purpose of each as it is used:

Character	Purpose
#	The hash character designates data fields in Screen Design. May be user-defined.
“	Double quotes enclose a text literal. (Can vary in some countries.)
‘	A single quote (apostrophe) marks a continuing line in programming mode.
( )	Parentheses appear in arithmetic or text expressions and in the FILE, SCREEN, PROGRAM, ACCESS, VIEW, INTERFACE, and other library statements for naming conventions.
:	A colon separates two programming statements on the same line. A colon can also separate a library from an entity name.
;	A semicolon indicates the suppression of tabbing on an unformatted screen. See “SHOW” on page 400 for an explanation.
,	A comma separates parameters and subscripts, and indicates tabbing on an unformatted screen.
.	A period designates a decimal point in a number. (It is the default decimal point for screen fields, which can be changed by users.)
—	An underline connects two or more words in a symbolic name.

Character	Purpose
	A vertical bar marks a comment line in programming mode. In Screen Design, it is the default blank-fill character. It can be used to tie together fields (e.g., words in a heading) into a single field or to indicate automatic skipping (tabbing) between fields.
!	An exclamation point marks a Double Byte Character String (DBCS) comment.
+	A plus sign adds two numeric data items or concatenates two character data items.
-	A minus sign subtracts two numeric data items or deconcatenates two character data items. Special note: Do not use a minus sign or dash between two words in a file name (e.g., <i>file-name</i> ), or MANTIS tries to subtract the values.
*	An asterisk multiplies two data items.
**	A double asterisk raises one number to the power of the second number.
/	A slash divides one number by the value of the second number.
=	An equal sign evaluates an expression to TRUE if both sides are equal; otherwise, it evaluates to FALSE. In a LET statement an equal sign sets the expression(s) on the left hand side to the expression(s) on the right hand side. This character is also used for the ATTRIBUTE, and PRINTER, DATE, and TIME functions (that is the LET.)
@	The at sign is used by the Component Engineering Facility to recognize a source program when found as suffix to a MANTIS program name. Also used to <i>nominat</i> e a component to be broken out in the Decompose process. May be user-defined.

## Symbolic names

A symbolic name is a string of characters that represents a user-defined object (such as a screen or field) in a MANTIS program. MANTIS uses symbolic names to represent variables processed by a MANTIS program. Symbolic names can stand for either numeric or text data. MANTIS allows a maximum of 2048 symbolic names for a single program, including names defined indirectly by SCREEN, FILE, and ACCESS statements.

A symbolic name:

- ◆ Must begin with an alphabetic character.
- ◆ Can contain alphabetic characters, numeric characters, and the underline (\_). No other special characters are allowed in a symbolic name. Lowercase characters can be entered; MANTIS will convert them to uppercase (e.g., the following variables are equivalent: `customer_name`, `Customer_Name`, `CUSTOMER_NAME`).
- ◆ Must not be a reserved word, as listed later in this section. A symbolic name can contain a reserved word (e.g., `EDITOR`), but cannot be a reserved word in its entirety (e.g., `EDIT`).
- ◆ Can be any size that fits on a line. However, if the field is used in a design entity, (e.g., screens, interfaces, or files) it is limited to 16 or 30 characters.
- ◆ When MANTIS executes an ACCESS, FILE, INTERFACE, SCREEN, TOTAL, or VIEW statement (a *complex* statement), MANTIS defines the symbolic name specified in the statement. MANTIS also defines all of the as-yet undefined fields that you have in the design of that object.
- ◆ Must be unique. When a symbolic name is previously defined, MANTIS bypasses the subsequent definition. However, with complex statements such as ACCESS, MANTIS checks the subfield's current and new datatype for consistency.

MANTIS automatically converts all symbolic names and reserved words to uppercase in your program.

MANTIS reserves certain words for command names, built-in functions, and other features of the language.

If you use the PREFIX option on a complex statement, the variable name will be appended to the complex entity name to form a symbolic name.



The following is a list of MANTIS reserved words:

ABS	AFTER	ALTER	ASI	ATN
ACCESS	ALL	AND	AT	ATTRIBUTE
BEFORE	BIG	BIND	BLOB **	BREAK
BY				
CALL	CHANGE	CLEAR	CONVERSE	COS
CHAIN	CHR	COMMIT	COPY	CURSOR
DATAFREE	*DBCS	DECIMAL **	DELETE	DISPLAY
DOLEVEL	DATE	DBPAGE **	DEQUEUE	DO
DOWN				
E	END	EQUAL	EXEC_SQL	EXIT
EDIT	ENQUEUE	ERASE	EXECUTE	EXP
ELSE	ENTRY			
FALSE	FIRST	FOR	FORMAT	FSI
FILE				
G"	GET	GO **		
HEAD	HELP			
IF	INSERT	INT	INTEGER **	INTERFACE
INTERNAL				
K"	KANJI	KEY		
LANGUAGE	LET	LIST	LOG	LOWERCASE
LAST	LEVEL	LOAD	LUID	
MARK	MIXD	MIXMODE	MIXT	MODIFIED
MEMORY	MIXM			

NEW	NEXT	NOT	NULL	NUMERIC
OBTAIN	ON	OR	ORD	OUTPUT
OFF				
PAD	PERM **	POSITION	PRIOR	PROMPT
PASSWORD	PI	PREFIX	PROGFREE	PURGE
PERFORM	POINT	PRINTER	PROGRAM	
QUIT				
RELEASE	RESET	RND	ROUNDING	RUN
REPLACE	RETURN	ROUNDED		
SAME	SELECT	SHOW	SLOT	SQLDA
SAVE	SEQUENCE	SIN	SMALL	SQR
SCREEN	SET	SIZE	SQLBIND	STOP
SCROLL	SGN	SLICE	SQLCA	SUBMIT
SEED				
TAN	TERMSIZE	TIME	TOTAL	TRUE
TERMINAL	TEXT	TO	TRAP	TXT
ULTRA	UNTIL	UPDATE	USAGE	USERWORDS
UNPAD	UP	UPPERCASE	USER	
VALUE	VIA	VIEW	VSI	
WAIT	WHEN	WHILE	WINDOW	

ZERO

\*\* Reserved for future use

---

## Numeric considerations

The MANTIS number set consists of:

- ◆ Digits 0–9
- ◆ Preceding plus or minus sign
- ◆ Period
- ◆ Letter E

Internally, MANTIS stores numeric data in floating point and regards numbers in one of two ways:

- ◆ **SMALL.** Stores a four-byte floating-point number.
- ◆ **BIG.** Stores an eight-byte floating-point number.



---

Even if your installation uses a decimal point other than the period (.), for example the comma (,) in user screens, you must use a period (.) for a placeholder in the numbers in your programs.

---

You can also store numeric data in arrays. Arrays are ordered sets of numbers that have one or two dimensions. You can specify arrays as BIG or SMALL. For example, if you specify a 1-dimensional array:

```
BIG DATA(5)
```

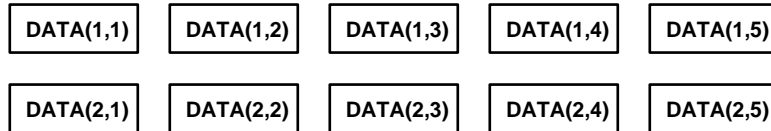
MANTIS allocates storage for five BIG occurrences of DATA:



If you specify a 2-dimensional array of two rows and five columns:

```
BIG DATA(2,5)
```

MANTIS allocates storage as (sometimes called “row major”):



See “[MANTIS conventions](#)” on page 31 for more detailed information on scientific notation and numeric arrays.

You can also store text data in a 1-dimensional array using only the TEXT statement. For example, if you specify:

```
10 TEXT DATA(3, 10)
```

MANTIS allocates storage as:



where each DATA element has 10 bytes (characters) of storage available



The following screen illustration shows the standard facilities menu provided with MANTIS. Your Master User may have omitted some of these facilities, and/or added new facilities to meet your specific needs. To access a facility from the menu, enter the number of the facility in the selection field, and press ENTER.

FAC002	MANTIS Facility Selection Menu		YYYY:MM:DD
	user		HH:MM:SS
Please select one of the menu options below.			
—	Run a Program by Name .....	1	Sign On as Another User .... 11
	Display a Prompter .....	2	Search Facility .....
	Design a Program .....	3	Query Report Writer .....
	Design a Screen .....	4	Directory Facility .....
	Design a MANTIS File View ..	5	Transfer Facility .....
	Design a Prompter .....	6	Cross Reference Facility ... 16
	Design an Interface .....	7	Entity Transformers .....
	Design a TOTAL File View ...	8	Universal Export Facility .. 18
	Design an External File View	9	Print Facility .....
	DL/I Access View .....	10	
F1=HELP F3=END F12=CANCEL			

For information on using the Program Design Facility and the Full Screen Editor (FSE) to create and update programs, refer to *MANTIS Program Design and Editing, OS/390, VSE/ESA*, P39-5013. For a discussion of MANTIS programming conventions, see “MANTIS conventions” on page 31.

# 2

## MANTIS conventions

This chapter discusses MANTIS language conventions. The language fundamentals discussed in “[Overview of MANTIS language](#)” on page 21 also apply here. It does not attempt to teach you how to program, but rather outlines MANTIS basics which you can apply using your current data processing knowledge. Short program sequences illustrate specific aspects of MANTIS programming. You need not understand each statement at this point. “[MANTIS programming language](#)” on page 75 provides details on MANTIS commands, functions, and statements.

### Programming fundamentals

The three basic elements of MANTIS programming mode are as follows:

- ◆ **Statements.** Part of a program; requires a run-mode action—they are not executed until the program is run (e.g., FILE, SCREEN). MANTIS statements require a line number. MANTIS automatically indents statements to indicate their relative position in the program’s nesting hierarchy.
- ◆ **Commands.** Not a part of a program, but designate an immediate-mode action—they are executed immediately (e.g., LIST, RUN). Immediate-mode commands cannot have a line number.
- ◆ **Editing commands.** Part of the Full Screen Editor (FSE) and are used to maintain MANTIS programs. For more information on using the FSE refer to [MANTIS Program Design and Editing, OS/390, VSE/ESA](#), P39-5013.

Some of the MANTIS reserved words can be used as both statements and commands (such as EXIT). Most MANTIS statements can be entered without a line number and executed immediately (such as SHOW and LET).

You don't need to remember which reserved words are statements and which are commands. MANTIS returns an error message if you omit a line number on a statement or try to enter a line number on a command.

When you write a MANTIS program, you can use either the Full Screen Editor or the Line Editor (see Note below). The Full Screen Editor provides the entire screen (254 columns wide) for creating and maintaining programs. The Line Editor provides a line at the bottom of the screen for you to enter program statements and commands. MANTIS scrolls previously entered lines up and repositions the cursor on the bottom line each time you press ENTER.

You can run a partial or completed program at any point by entering the RUN command. Your program executes from the first to the last coded statement. MANTIS interprets programs as they run; with no compiler and no object code. MANTIS first checks the logical structure of your program to ensure matched statements, such as IF-END, ENTRY-EXIT.

You can save a program that is not logically complete (such as missing an END statement), but the program must be logically balanced to RUN it. Refer to the SAVE command in *MANTIS Facilities, OS/390, VSE/ESA*, P39-5001.

If MANTIS encounters an error in a running application, it returns the name of the program as well as the line number where the error was found. MANTIS terminates a program when any of these occur:

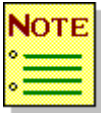
- ◆ MANTIS executes a STOP statement.
- ◆ MANTIS executes the EXIT associated with the ENTRY statement of the program.
- ◆ No more statements to process (physical end of program).
- ◆ MANTIS encounters an error.
- ◆ The KILL command (or its equivalent as specified by the Master User) is entered in response to terminal input.

Refer to *MANTIS Messages and Codes, OS/390, VSE/ESA*, P39-5004, for warnings and error messages you might receive in the programming process.



## Automatic mapping

Automatic mapping is the process that MANTIS uses to allow the sharing of data areas between variables of like name and data type. For example, CUST\_NAME defined as a text field found on a screen can share the same memory location as a text field in a file called CUST\_NAME. This mapping occurs automatically whenever an already defined name is encountered. We recommend that you use standard naming conventions on a system-wide basis to make the best use of this feature.



---

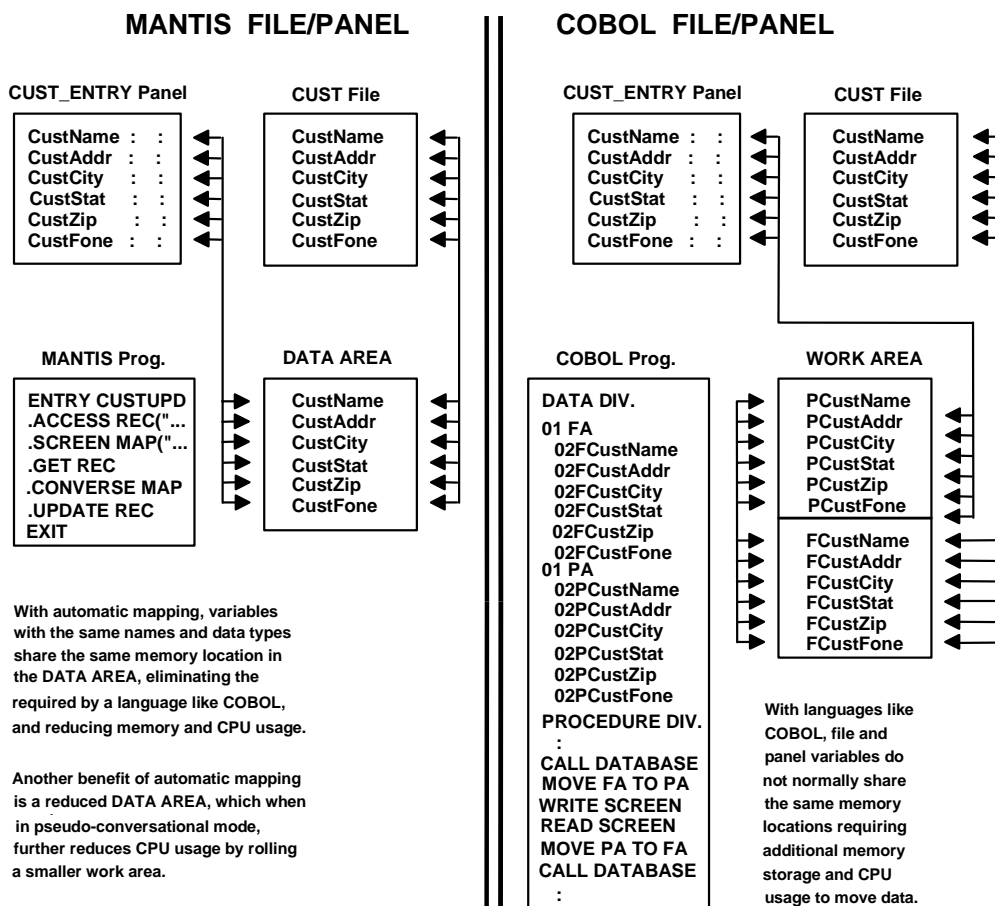
Using the PREFIX parameter in the FILE, ACCESS, and TOTAL statements inhibits automatic mapping. See “[FILE](#)” on page 222 and “[Automatic mapping](#)” on page 541 for more information.

---

### Advantages of automatic mapping

If you have programs with large numbers of LET statements to move fields between different areas (e.g., screens and files), the use of automatic mapping can reduce the complexity and chance for errors in your program, and also reduce the CPU usage and memory requirements of your program. With many languages such as COBOL, the programmer defines the memory locations for screen and file variables separately. When data is read, moved, and written, there can be a considerable use of memory and CPU usage. This is not the case with MANTIS. Automatic mapping saves coding time, memory, and CPU usage.

The following figure shows a comparison between COBOL and MANTIS:



**Note: COBOL example is in pseudo-code.**

The advantage of using automatic mapping can be lessened if a standard naming convention is not used throughout the shop. Since MANTIS does duplicate definition checking only in certain situations (such as data type), naming conventions should also include standards in text lengths. If this convention is not followed, it is possible to have a field defined with two different lengths. For example, it is possible to define a text field as 55 bytes long on a screen, and 35 bytes long in an ACCESS view. MANTIS uses the first field length encountered when the program is run. That is, if MANTIS encounters the ACCESS statement first, the data area is set up with a length of 35 bytes for the field.

Most shops are database driven, and should always specify ACCESS, VIEW, TOTAL, and FILE statements first in a MANTIS program to ensure that the MANTIS program always uses the most current definition for a field.

Another AD/Advantage product, XREF, can be used to check variables to ensure standard naming, data type, and length consistency. Consistency checking when setting up the data area allows you to use automatic mapping to your best advantage.

## Statements

A program statement consists of a line number (1 through 30000), a MANTIS reserved word, and 0 or more operands, depending on the statement. You can use spaces freely to make statements easier to read. MANTIS ignores spaces except when they appear in text literals and comment statements, or perform a delimiting function. For example, the following two statements have the same meaning to MANTIS:

```
10  SHOW  ALPHA,   BETA,  - 1500+ 7
20  SHOW  ALPHA,BETA,    -1500  + 7
```

If you list these statements, MANTIS removes all unnecessary blanks and inserts necessary blanks, as follows:

```
LIST
10  SHOW ALPHA,BETA,-1500+7
20  SHOW ALPHA,BETA,-1500+7
```

When you list a program, periods (.) can appear between the statement number and the statement. If you set your User Profile to Indent On in the Full Screen Editor (FSE), MANTIS inserts these periods *automatically* to show your program's nesting hierarchy. You need *not* supply periods when you enter statements.

```
LIST
15  ....SHOW ALPHA,BETA,-1500+7
16  ....SHOW ALPHA,BETA,-1500+7
```

Program lines created using the FSE can contain up to 254 characters. Use the FSE scrolling function to move your terminal window to the right and continue your program line past your current terminal width. With the Line Editor, you can create program lines as wide as your terminal. (See note below regarding Line Editor usage.)

In either mode, if you need more characters on the line than the screen holds, continue a line by entering an apostrophe (') immediately after the next line number. When you press ENTER, MANTIS displays the line with a space between the line number and the apostrophe.

```
10  SHOW "THE RELATIVE HUMIDITY IS " ;HMD; "AND THE TEMPERATURE IS ";
20  'TEMP
```

Do not break up a reserved word, a symbolic name (for example, TEMP), or a number (for example, 12345.67). Both of the following continuation attempts are *incorrect*:

```
10 SHOW "THE RELATIVE HUMIDITY IS ";HMD;"AND THE TEMPERATURE IS " ; TE
20 'MP
30 SHOW "THE RELATIVE HUMIDITY IS ";HMD;"AND THE POLLEN COUNT IS " ;123
40 '45.67
```

If you are using the Line Editor and a listed line's length or indentation cause it to exceed your terminal width, MANTIS shows a plus sign (+) as the character right after the line number. The plus sign (+) can also appear when you create a long line in the FSE and then list the program in the Line Editor. When the plus sign appears in this way, it has nothing to do with nesting.

To display the rest of the line, you may be able use the ALTER command. Lines that exceed 72 characters will not be able to be listed or altered in the line editor. These lines can be created and changed in the Full Screen Editor, by scrolling to the right to view the rest of the line. In the Full Screen Editor, MANTIS does not display a plus sign.

If you have a long literal (text enclosed in quotes) and want the entire line to show on a single screen, close the literal on the first line (closing ") and reopen it (opening ") on the continuation line:

```
60 SHOW "THIS IS THE WEATHER REPORT FOR ALL REGIONS SOUTH AND "
70 ' "SOUTHWEST OF CHICAGO"
```

With the Full Screen Editor, you can also scroll to the right and continue the line, or use the procedure previously described.



Cincom recommends that you do **not** use the Line Editor (available with earlier MANTIS versions) with the Program Design Facility. The Line Editor does not update the Extended Entity Profile Records (EEPR) of the program, needed in order for the Program Design Facility to function properly. When you use the SAVE, REPLACE, DELETE, and BIND commands with the Line Editor, program integrity can be affected. Each time you access the Line Editor, you receive a message warning you of this possibility. For information on using the Line Editor, refer to *MANTIS Program Design and Editing, OS/390, VSE/ESA, P39-5013*.

You can include two or more statements on the same line by separating them with a colon (:).

```
90 CLEAR:HEAD "THIS LINE APPEARS AT THE TOP OF YOUR SCREEN"
```

You *cannot*, however, include the following MANTIS verbs on a line with another statement:

BREAK	ENTRY	IF	UNTIL
DO	EXIT	NEXT	WHEN
ELSE	FOR	RETURN	WHILE
END			

These statements (known as logic verbs) must appear on a line by themselves.

## Commands

A MANTIS command is a reserved word *without* a line number. Thus, MANTIS does not consider a command to be part of your program and executes it immediately. For example, you use commands to RUN or LIST a program. You can execute some statements as commands by entering them without a statement number. Again, such a statement is not part of the coded program and MANTIS executes it immediately. In the following example, SHOW appears as both a statement and a command.

### SHOW as a statement

In this example, SHOW appears as a statement:

```
EDIT L1 --- EXAMPLES:COMPOUND                                COLUMNS 1 73
COMMAND ==> run                                              SCROLL ==> CUR
***** ***** START OF PROGRAM *****
00010 ENTRY COMPOUND
00020 .SHOW"WHAT IS THE CAPITAL AMOUNT?"
00030 .OBTAIN INVESTMENT
00040 EXIT
***** ***** END OF PROGRAM *****
```

This results in:

```
WHAT IS THE CAPITAL AMOUNT?
```

```
1400
```

## SHOW as a command

In this example, SHOW appears as a command:

```

EDIT L1 --- 2050                                COLUMNS 1 73
COMMAND ==> show investment+650                SCROLL ==> CUR
***** START OF PROGRAM *****
00001 ENTRY COMPOUND
00002 .SHOW"WHAT IS THE CAPITAL AMOUNT?"
00003 .OBTAIN INVESTMENT
00004 EXIT
***** END OF PROGRAM *****

```

## Comments

You can insert comments among your program statements by entering a vertical bar ( | ) before your comment.

```

10  ENTRY COMPOUND
20  . | THIS IS AN EXAMPLE
30  .SHOW"WHAT IS THE CAPITAL AMOUNT?"
40  .OBTAIN INVESTMENT
50  EXIT

```

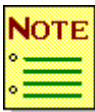
You can also include comments on the same line with a statement by entering a colon (:), a vertical bar (|), and your comment after the statement:

```

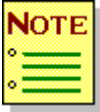
10  ENTRY COMPOUND: | THIS IS AN EXAMPLE

```

You can use all characters in the MANTIS character set in your comments. Because MANTIS ignores comments when executing a program except for SQL statement text, don't put statements or commands on the same line after a comment.



DBCS (Asian language support) comments are indicated by an exclamation point (!). If you make a keying error and type an exclamation point in your program line, MANTIS returns an error message.



---

There are two special comment forms that affect spacing when printing a program in the MANTIS Print Facility. Insert the following comments into your program wherever you would like to add extra blank lines while printing the program:

- ◆ | **EJECT**. Causes the print program to skip to the top of a new page.
  - ◆ | **SKIP**  $\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$ . Causes the print program to skip 1, 2, or 3 lines.
-



---

## Numeric data

This section discusses the arithmetic considerations MANTIS uses in its storage and manipulation of numeric data.

MANTIS stores numeric data in hexadecimal floating point. However, you normally see data displayed in standard numeric notation. In program lines, or if you use an unformatted screen and have a number greater than 15 digits, MANTIS displays that number in scientific notation. Your Master User may change the digit threshold at which numbers display in E-notation (scientific notation).

Numeric variables defined in a program have an initial value of 0 (except those passed to an ENTRY statement.)

### Numeric literals and variables

MANTIS stores numerics as either literals or variables.

Numeric literals, or constants, can appear in your program as valid MANTIS numbers; for example, 4, 1.43, 0.07, or 14E7. You may enter numeric literals in a number of ways, but MANTIS will redisplay them in its standard way—for example, on SHOW or in a program LIST. Numeric variables can contain numeric values that can be reassigned during program execution. Examples of numeric variables and literals are:

```
X=4
Y=1.43
Z=0.07
I=X
```

You can define a numeric variable in a program in the following ways:

- ◆ With a BIG or SMALL statement (see “[Scientific notation \(E-notation\)](#)” on page 42). The variable is initially 0.
- ◆ With numeric field definitions in a SCREEN, FILE, INTERFACE, TOTAL, ACCESS, or VIEW statement. The variable is initially 0.

- ◆ As a parameter on the program's main entry statement where the argument passed was BIG or SMALL. The variable has the BIG or SMALL definition and the value of the invoking program's parameter.
- ◆ With an as-yet undefined variable in a statement. It will default to a BIG with a 0 value. For example, the following two code samples are the same:

- Code sample 1:

```
10 BIG X,Y
20 X=Y+3
```

- Code sample 2:

```
10 X=Y+3
```

## Scientific notation (E-notation)

MANTIS accepts 0 or any number with magnitude 1E-74 through 9E+73. The letter E, in scientific notation, means “times 10 to the power of.” The number after the E specifies how many places the decimal point shifts (to the right if positive, to the left if negative). The number following the E must be an integer. For example, you can write:

```
100 as .1E3           1.7640 as .1764E1
23000 as .123E5       .0004 as .4E-3
```

MANTIS mayabend if a numeric statement causes an underflow or overflow of the system's floating-point limit.

MANTIS stores numbers as floating point native to the machine. The nature of floating point results in some numbers being approximated. As mentioned briefly in “[Overview of MANTIS language](#)” on page 21, MANTIS stores numbers in one of the following ways:

- ◆ **SMALL.** SMALL variables can accurately represent integers from -16777216 to +16777216 (about 1e7). Use SMALL for whole numbers within this range.
- ◆ **BIG.** BIG variables can accurately represent integers from -72057594037927936 to +72057594037927936 (about 7e16). Use BIG to approximate any number containing fractions and whole numbers whose magnitude exceeds 1e7.

Round-off errors can occur in calculations involving BIG and SMALL numbers. Using the **ROUNDED** option will help maintain accuracy in calculations.

Follow the convention of writing numbers in scientific notation. To do so, remove all leading and trailing zeroes, then move the decimal point to the left if a whole number or to the right if a fraction. The mantissa (the part preceding the exponent E) must be between -1.0 and 1.0.

The following table gives several examples of E notation:

Original number	E notation	Comments
12	.12E2	
1230000000	.123E10	
-123E7	-.123E10	
123E7	.123E10	
.001	.1E-2	
123.456	.123456E-3	

Consider the following MANTIS program:

```

10  SMALL ALPHA,BETA
20  ALPHA=12300000000
30  BETA=1234567890

```

In statement 10, ALPHA and BETA are variables with a SMALL significance. Both assignments will yield incorrect values because the values exceed the capacity for a SMALL.

In such cases, use BIG to define your variables:

```

10  BIG ALPHA,BETA
20  ALPHA=12300000000
30  BETA=1234567890

```

You would not lose precision because a BIG can hold these numbers accurately. When you write a program, remember to specify the degree of precision your variables need by specifying either SMALL or BIG. If you don't specify either, MANTIS assumes BIG.

Because MANTIS stores numeric data in floating point, specify the number of decimal digits that you want carried out when you perform a calculation that results in a real number. For more details, see the **ROUNDED** option of the LET statement.

For very large numbers or fractions there can be some loss of precision based on the way the number is stored in the machine's native hexadecimal floating point. Arithmetic operations, especially when repeated, can accumulate precision-related differences in floating point numbers.

## Arithmetic arrays

An arithmetic array is an ordered set of numeric values. Each value is called an array element. Arithmetic arrays can have one or two dimensions. You can define arithmetic arrays with BIG or SMALL statements, or with such statements as SCREEN, FILE, INTERFACE, VIEW, TOTAL, or ACCESS if these definitions contain implied arrays.

[“Overview of MANTIS language”](#) on page 21 discussed how MANTIS allocates memory to hold array elements. To access one element of an array, specify its position within the array by subscripting the variable. A subscript can be either a number or an arithmetic expression. For example, if you have an array with 17 elements:

```
10 BIG ALPHA(17)
```

You can access an element within an array by entering  $\text{ALPHA}(n)$ , where ALPHA is the variable name and  $(n)$  is the subscript notation. To access the eleventh element in an array, use any of the following notations:

```
ALPHA(11) or  
K=11 : ALPHA(K) or  
ALPHA(6+5) or  
ALPHA(1)=9 : K=2 : ALPHA(ALPHA(1)+K)
```

The value you use to subscript an array must lie between 1 and the maximum number of elements defined for the array (inclusive). If the value lies outside this range, MANTIS returns a subscript out of range error message. If, for example, you specify:

```
10  SMALL BETA(7,3)
20  BETA(6,4)=2
```

MANTIS issues an error message at statement 20 because the second dimension (4) in statement 20 exceeds the maximum (3) defined for the second dimension in statement 10.

Arithmetic expressions

An arithmetic expression in MANTIS consists of operands and operators. You must separate each pair of operands with one operator. The following table lists valid operands and an example of each type. The next table then describes the valid operators that you use to separate the operands described below.

Operands	Example
Arithmetic Variables	ZEBRA, BUCK
Array Elements	A(5), C(2), X(Y+Z)
Built-in Functions	SIN(Y), NOT(Z)
Numeric Constants	1.43, 0.07, 14E-7
Built-in Constants	PI, E, FALSE, ZERO
Arithmetic Expressions in parentheses	(4+SIN(Y))
Logic Expressions in parentheses	(A = B)

An operator is a symbol with an arithmetic or logical function (e.g., +).

The following table contains all valid MANTIS numeric operators, a brief description of their functions, and an example of each. In each example, A and B represent operands. The preceding table outlines the valid types of operands. Each pair of operands from the preceding table must be separated by a valid operator from the following table. That is, you must separate operands by operators, and you must separate operators by operands or parentheses.

Symbol	Meaning	Example
+ or - (leading)	Unary sign of A.	+A or -A
+	Add A to B.	A + B
-	Subtract B from A.	A - B
**	Raise A to the power B.	A ** B
*	Multiply A by B.	A * B
/	Divide A by B. Note that if B is equal to zero, MANTIS sets the result to zero to avoid the error condition when dividing by zero.	A / B
=	If A and B have the same value, the expression evaluates to TRUE, that is (1); otherwise, it evaluates to FALSE, that is, (0). Also indicates the assignment of a value to a variable (e.g., LET X=Y).	A = B

Symbol	Meaning	Example
<	If the value of A is less than the value of B, the expression evaluates to TRUE; otherwise, it evaluates to FALSE.	A < B
>	If the value of A is greater than the value of B, the expression evaluates to TRUE; otherwise, it evaluates to FALSE.	A > B
<=	If the value of A is less than or equal to the value of B, the expression evaluates to TRUE; otherwise, it evaluates to FALSE.	A <= B
>=	If the value of A is greater than or equal to the value of B, the expression evaluates to TRUE; otherwise, it evaluates to FALSE.	A >= B
<>	If the value of A does not equal the value of B, the expression evaluates to TRUE; otherwise, it evaluates to FALSE.	A <> B
AND	If both operands are TRUE (that is, nonzero) the expression evaluates to TRUE; otherwise, it evaluates to FALSE.	A AND B (A=B) AND (C=D)
OR	If either operand (or both) is TRUE (nonzero), the expression evaluates to TRUE; otherwise, it evaluates to FALSE.	(A=B) OR (C=D) FLAG1 OR FLAG2

The MANTIS programming language uses explicit operators, unlike algebra, which uses implicit operators. For example:

Algebraic Expression	MANTIS Expression	Comments
$2a+b$	$2 * A + B$	* is the multiplication operator.
$2(b+c)$	$2 * (B + C)$	
$\frac{A+B}{C+D}$	$(A + B) / (C + D)$	Parentheses indicate that A+B and C+D are added before the division.
$b^2-4ac$	$B ** 2 - 4 * A * C$	** is exponentiation.
$\sqrt{b^2-4ac}$	$SQR (B ** 2 - 4 * A * C)$	Uses MANTIS function SQR (square root).

You can use the unary + and - operators only at the beginning of an arithmetic expression or immediately after an opening parenthesis. For example:

<code>-ALPHA+GAMMA</code>	<i>Valid</i>
<code>ALPHA/ (-GAMMA)</code>	<i>Valid</i>
<code>IF -ALPHA=GAMMA</code>	<i>Valid</i>
<code>ALPHA/ -GAMMA</code>	<i>Invalid—one operator immediately follows another</i>
<code>ALPHA+ -GAMMA</code>	<i>Invalid—one operator immediately follows another</i>
<code>IF ALPHA=-GAMMA</code>	<i>Invalid—one operator immediately follows another</i>

The unary + operator has no effect on the evaluation of the expression. The unary - operator changes the sign of the operand following it from positive to negative, or to positive if the sign of the operand is already negative.



MANTIS evaluates numeric and relational operators in the following order:

Operator Precedence	Description	Example
( )	Expressions in parentheses	(A+B)*C (the sum of A and B is multiplied by C)
UNARY (+, -)	Unary operators	-A
**	Exponentiation (to the power of)	A**B
* /	Multiply, divide	A/B
+ -	Add, subtract	A-B
= < > >= <= <>	Relational operators	A<>B
AND	Conjunction	A AND B
OR	Disjunction	A OR B

When two operators have equal priority (for example, addition and subtraction), and there are no parentheses, MANTIS evaluates the expression from left to right.

You can add parentheses even when they are not needed, if you think they make the expression clearer.

Use parentheses in arithmetic expressions to enclose subexpressions that function as entities. MANTIS evaluates a subexpression in parentheses first. For example, if you enter:

```
YEAR + DAY / ( 6 * (MONTH-LAG) )
```

MANTIS first subtracts LAG from MONTH, multiplies that result by 6, then divides DAY by that new result, and finally adds YEAR to the result of the division.

## Text data

This section discusses the text considerations MANTIS uses when storing and manipulating text data.

MANTIS stores text data in ordered character strings, each with an associated current and maximum length. The maximum length can range from 1 to 254 characters. The current length can range from zero to the maximum length. Text data can include bytes of any hexadecimal value. See “**MIXMODE**” on page 329 for exceptions.

Text variables defined in a program have an initial current length of zero. The exception is those variables that are passed to an ENTRY statement parameter.

### Text literals and variables

MANTIS stores text as either literals or variables. Text literals and variables can contain uppercase or lowercase letters, symbols, and so on (they can contain any hexadecimal value).

A text literal is any set of 0–254 MANTIS characters enclosed in quotes; for example, “THIS IS A TEXT LITERAL”. Use two consecutive quotes to specify one occurrence of a quote mark within a text literal. For example, if you run the following:

```
10 SHOW "THE ANSWER IS X", "THE ANSWER IS "Y" "
```

MANTIS returns:

```
THE ANSWER IS X          THE ANSWER IS "Y"
```



---

The symbol used for the quote mark is different in some countries. If you have trouble, consult your Master User for the proper character to use for the quotes.

---

## Defining text variables

You can define a text variable in a program in the following ways:

- ◆ With a TEXT statement. The variable is initially a zero-length string. The default maximum length (if you do not specify a dimension on the TEXT statement) is sixteen.
- ◆ With text definitions within a SCREEN, FILE, INTERFACE, TOTAL, ACCESS, or VIEW statement. The variable is initially a zero-length string.
- ◆ As a parameter on the program's main entry statement where the argument passed was TEXT. The variable has the TEXT definition and the actual data length of the invoking program's parameter.

The default for a variable is numeric (see BIG), so you *must* define text variables before using them. If you specify:

```
10 TEXT DATA(3)
```

MANTIS allocates storage for a 3-character text variable:

**DATA**

--	--	--

TEXT variables can be treated as arrays of 1 byte each. This provides for substrings. DATA(2,2) references the second through the second character of DATA (for a length of 1).

If you specify TEXT DATA(2,4), MANTIS allocates storage for a list of two variables that can contain up to four characters each:

**DATA(1)**

**DATA(2)**


A text variable cannot exceed 254 characters in length, while an array of text variables cannot contain more than 255 entries.

When a text variable is defined, it has a current length of 0 (zero). A text variable has the following characteristics:

- ◆ A *defined length* (or maximum length) as specified in the defining statement.
- ◆ A *current length* maintained by MANTIS. The current length indicates how many characters the text variable currently contains.

See the following examples:

Example	Results	Comments
TEXT ALPHA ( 20 )	ALPHA = " "	MANTIS creates a field, ALPHA, with a maximum length of 20 characters and a current length of 0 characters.
ALPHA = "ITEM NUMBER "	"ITEM NUMBER "	ALPHA has a current length of 11 characters.
SIZE ( ALPHA )	11	Current length.
SIZE ( ALPHA , "MAX" )	20	Defined (maximum) length.
TEXT BETA ( 10 )		MANTIS
BETA = "12345678901234567890 "	"1234567890 "	truncates the expression to the variable's defined length.

### Substringing text variables

You can use subscripts to reference portions of a text variable. If, for example, you enter:

```
10 TEXT MESSAGE(50)
20 MESSAGE="CUSTOMER NOT FOUND"
30 |          123456789012345678           for position reference
```

then:

Example	Results	Comments
MESSAGE	"CUSTOMER NOT FOUND"	Current length is 18 characters.
MESSAGE(10)	"NOT FOUND"	Position 10 through the end of the current length.
MESSAGE(10,12)	"NOT"	Positions 10 through 12 of the string.
MESSAGE(19)	" "	Current length is 18 characters.

MANTIS also uses negative subscripts to refer to the position within the text variable, but references from the end of the current length of the variable (when MESSAGE="CUSTOMER NOT FOUND"):

Example	Results	Comments
MESSAGE(-1)	"D"	The last character.
MESSAGE(-5)	"FOUND"	The fifth-to-last character through the end of the current length.
MESSAGE(-9,-7)	"NOT"	The ninth through seventh characters from the end of the string.

If you refer to an array of text variables, the first subscript nominates the entry; that is, if:

```
TEXT MESSAGE(6,16)
MESSAGE(6)="DUPLICATE RECORD"
```

then:

Example	Results	Comments
MESSAGE(6,1,3)	"DUP"	The sixth array element, from positions 1 through 3.
MESSAGE(6,-6,-2)	"RECOR"	The sixth array element, from the sixth-to-last character up to the second-to-last character.

A subscript can also be an arithmetic expression. If, for example, you enter:

```
10 TEXT MESSAGE(6,16)
20 MESSAGE(6)="UPDATE COMPLETE"
30 SHOW MESSAGE ((3*2),1,(1+5))
```

Line 30 goes through the following internal computational steps:

```
SHOW MESSAGE((3*2),1,1+5)  that is equivalent
SHOW MESSAGE(6,1,6)        to the text string:  UPDATE
```

You can also substring a text variable on the left of an equal sign in an assignment (LET) statement. See “[LET \(TEXT/KANJI/DBCS variables\)](#)” on page 312. PAD and UNPAD statements can also operate on TEXT variable substrings.

## Text expressions

A text expression in MANTIS consists of operands and operators. You must split up each pair of operands with one operator. The following table lists valid operands and an example of each type. The next table then describes the valid operators that you use to separate the operands described below.

Operands	Examples
Text variables	GIRAFFE, DOE
Text variable substrings	GIRAFFE(3,6), DOE(7), DOE(-4,-2)
Array elements	T(5), W(2), U(Y+Z)
Built-in functions	KEY, PASSWORD, POINT(S-"@" ), FORMAT(X,MASK)
Text constants (literals)	"END", "Press ENTER"
Built-in constant	NULL
Text expressions in parentheses	(QUESTION+" ?")

The following table contains all valid MANTIS text operators, a brief description of their functions, and an example of each. In each example, A and B represent operands. The preceding table outlines the valid types of operands. Each pair of operands from the preceding table must be split up by a valid operator from the following table. That is, you must separate operands by operators, and you must separate operators by operands or parentheses:

Symbol	Meaning	Example
+	Add A to B. (Concatenate the text strings.)	A + B
-	Subtract B from A. (Remove the <i>first</i> occurrence of the string in B—if there <i>are</i> any occurrences—from the string in A.)	A - B
=	If A and B have the same value, the expression evaluates to TRUE—that is, (1); otherwise, it evaluates to FALSE—that is, (0). The same value indicates both the same current length and the same contents for that length. It also indicates the assignment of a value to a variable (for example., LET X=Y). See the first note under the “Relational text expressions” heading on page 58.	A = B
<	If the value of A is less than the value of B, the expression evaluates to TRUE; otherwise, it evaluates to FALSE.	A < B
>	If the value of A is greater than the value of B, the expression evaluates to TRUE; otherwise, it evaluates to FALSE.	A > B
<=	If the value of A is less than or equal to the value of B, the expression evaluates to TRUE; otherwise, it evaluates to FALSE.	A <= B
>=	If the value of A is greater than or equal to the value of B, the expression evaluates to TRUE; otherwise, it evaluates to FALSE.	A >= B
<>	If the value of A does not equal the value of B, the expression evaluates to TRUE; otherwise, it evaluates to FALSE.	A <> B



Text addition (concatenation) and subtraction operators

In text addition (concatenation, the + operator) the right-hand operand is concatenated at the position following the current length of the first operand. When a text variable is assigned to an expression (for example, in a LET Statement), MANTIS truncates the part of the expression that exceeds the variable's maximum length.

In text subtraction (extraction, the - operator), MANTIS removes only the first occurrence of a string. You can repeat the operation to remove multiple strings or multiple occurrences. If the first operand does not contain the entire value of the second operand, the result is the entire first operand; no partial subtraction takes place. For example:

```
TEXT A(20),B(10)
```

Example	Results	Comments
A="ACTIVE"	"ACTIVE"	
B="IN"+A	"INACTIVE"	
A="INACTIVE"	"INACTIVE"	
B="PENDING"	"PENDING"	
A=A+" ":"+B	"INACTIVE:PENDING"	
A="INACTIVE"	"INACTIVE"	Truncated to 10 (max size of B).
B=A+"1234567890"	"INACTIVE12"	
A=NULL	" "	
B=A+"1234567890"	"1234567890"	
A="WEST MAIN AVE."	"WEST MAIN AVE."	A fills up 10 characters of B.
B=A+"1234567890"	"WEST MAIN "	
A="INACTIVE"	"INACTIVE"	Removes first occurrence of "I".
A=A-"I"	"NACTIVE"	
A="INACTIVE"	"INACTIVE"	Repeated subtraction in left- to-right order.
A=A-"I"-"IVE"	"NACT"	
A="ACCOUNT"	"ACCOUNT"	Entire "OO" string does not exist in A; no subtraction done.
B=A-"OO"	"ACCOUNT"	

Also see the PAD and UNPAD statements for adding and removing more than one occurrence of a character.

You can identify the presence of one string in another by using a text expression that returns a numeric value as the argument of the built-in text function POINT. If, for example:

```
TEXT VALID_KEYS
VALID_KEYS = "PF1 , PF2 , PF3 "
A=POINT ( VALID_KEYS- "PF3" )
B=POINT ( VALID_KEYS- "PF4" )
```

MANTIS sets A to a value of 8 because “PF3” *begins* in the eighth position of VALID\_KEYS. MANTIS sets B to zero because “PF4” does not appear in VALID\_KEYS.

Relational text expressions

A relational text expression will evaluate to a numeric: 0 if FALSE and 1 if TRUE. So you can mix the numeric and text expressions in valid ways; for example, to get K equal to 0 or 20, depending upon OPTION:

```
K=20 * ( OPTION= " YES " )
```

However, you cannot have a single operator between numeric and text operands. For example, “OK”+4 is invalid.

MANTIS evaluates text operators in the following order:

Operator precedence	Description	Examples
( )	Expressions in parentheses	A - ( B+C )
+, -	Concatenation and subtraction	A-B A+ " / "
=, >, <, >=, <=, <>	Relational operators	A=B A<>B

When two operators, such as addition and subtraction, have equal priority, and there are no parentheses, MANTIS evaluates the expression from left to right.

You can add parentheses even when they are not needed, if you think that they clarify the expression.

Use parentheses in text expressions to enclose subexpressions that function as entities. MANTIS evaluates a subexpression in parentheses first. For example, if you enter:

```
TEXT TRIAL,INVALID,EXCEPTIONS,OMIT
TRIAL="ABCD"
INVALID="CF"
EXCEPTIONS="C"
OMIT="A"
```

Example	Results	Comments
TRIAL+( INVALID-EXCEPTIONS )	"ABCDF"	Contents of EXCEPTIONS extracted from INVALID before the result is added to TRIAL.
TRIAL+INVALID-EXCEPTIONS	"ABDCF"	Operations performed in left-to-right order.
TRIAL+( INVALID-OMIT )	"ABCDCF"	Contents of OMIT is not in INVALID, so no extraction done.
TRIAL+INVALID-OMIT	"BCDCF"	Contents of TRIAL added to INVALID before the contents of OMIT extracted from the result.

Lowercase and uppercase letters do not compare equally. If you want to do a case insensitive compare, you can UPPERCASE (or LOWERCASE) both expressions, for example:

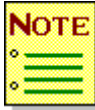
```
A = "Invoice"
B = "INVOICE"
```

Example	Results	Comments
A=B	FALSE (0)	Case sensitive
A<>B	TRUE (1)	Case sensitive
UPPERCASE(A) = UPPERCASE(B)	TRUE (1)	Case insensitive
LOWERCASE(A) = LOWERCASE(B)	TRUE (1)	Case insensitive



The comparison of text expressions depends on how your Master User has set up your system. Strings can be compared in the following ways:

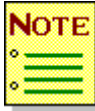
- 1. Length before content (default behavior).** Shorter strings compare lower. For example, "Z" < "AAA", since the first string is shorter. If the operands are of equal current length, they are compared, character-by-character, from left to right, until there is an inequality. For example, "ANNA" < "ANNE", because the E collates higher than the A in the last position.
- 2. Content before length.** For example, "Z" > "AAA" because Z follows A in collating sequence. If the strings compare equal up to the length of the shorter operand, the shorter operand compares less than the longer one. For example, "DAN" < "DANIEL".
- 3. By user exit.** This allows site-specific comparisons for special characters (for example, national language characters), case-insensitive comparisons, or comparisons irrespective of trailing blanks. For example, you may want "ö" to both compare equal to "oe" and collate between "o" and "p". This does not affect key order in files.



---

MANTIS truncates trailing blanks in input from VIEW, TOTAL, ACCESS, and INTERFACE files (external files). This can affect relational operations. Unless handled by the string compare exit, trailing blanks are significant; that is, "HI" <> "HI ".

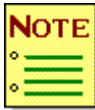
---



---

Hexadecimal data can be stored and handled but not necessarily displayed as intelligible data in a MANTIS text field.

---



---

Comparing nonalphabetic characters for anything other than equality may make your code nonportable to other MANTIS platforms, due to differences in EBCDIC vs. ASCII encoding and collating sequences. The same applies to accented and native language characters.

---

## DBCS considerations

MANTIS provides facilities to support the IBM DBCS character set. A DBCS character is a Double Byte Character Set (DBCS) character used with certain languages and on special terminals. DBCS data is represented (both in screen design and programming mode) with percent signs (%). DBCS characters can also be stored in TEXT variables with Shift-out and Shift-in characters (SO/SI) and must be enabled with the MIXMODE statement.

MANTIS stores DBCS data in ordered character strings, each with an associated current length. The length can range from zero to 127 characters (an even number of bytes, from zero to 254). Most operations work similarly to TEXT variables.

When you input DBCS data into an Internal Mixed field, use the (%) percent sign in the following ways:

- ◆ **Creation.** When you are creating the field, type in an even number of percent signs (%) representing the length of the field and press ENTER.
- ◆ **Extension.** If you have an existing field and you want to extend it, type in the percent signs (to make the total an even number) and a plus sign as follows: %%%%+ and then press ENTER.
- ◆ **Deletion.** If you have an existing field that is too long, and you want to eliminate some of the length, position the cursor on the first DBCS character you want to delete and press the EOF (end of field) key. MANTIS deletes the remaining characters in that field.

MANTIS recognizes the following special character sequences as DBCS designators in programming mode:

! Identifies a DBCS comment (instead of the vertical bar).

G“ ” Indicates a mixed literal.

K“ ” Designates a DBCS literal. Percent signs (%) represent the screen positions to be occupied by DBCS characters. The following is a sample DBCS literal: K“ %%%%%%%%% ”. MANTIS will convert the % fields to DBCS characters and redisplay the line for DBCS input.

## Built-in functions

MANTIS contains numeric and text functions that are used to return values within a program. The following table lists these functions along with a brief description. In the descriptions, *a* represents any arithmetic expression; *k* represents any DBCS expression; *t* represents any text expression. Each function is documented separately (with an example) in “[MANTIS programming language](#)” on page 75. The ASI, FSI, and VSI functions are documented in “[Status functions](#)” on page 517. The following table also describes the type of input and output used by each function and the type of function. An \* indicates that no input is needed by MANTIS.

Function	Description	Input	Output	Function
<a href="#">ABS(<i>a</i>)</a>	Returns the absolute value of <i>a</i> .	Numeric	Numeric	Mathematical
<a href="#">ASI</a>	Indicates the status of a field in a logical view.	Field name	Text	File Access
<a href="#">ATN(<i>a</i>)</a>	Returns the angle in radians whose tangent is <i>a</i> .	Numeric	Numeric	Mathematical
<a href="#">ATTRIBUTE</a>	Returns the current status of field, map, terminal, and printer attributes.	Name or Reserved word	Text	System
<a href="#">CHR(<i>a</i>)</a>	Returns a text value consisting of the character corresponding to the EBCDIC code specified.	Numeric	Text	String
<a href="#">COS(<i>a</i>)</a>	Returns the cosine of <i>a</i> where <i>a</i> is in radians.	Numeric	Numeric	Mathematical
<a href="#">CURSOR</a>	Indicates whether cursor appeared in a specific field at the last terminal I/O.	Field-name	True/False	System

Function	Description	Input	Output	Function
DATAFREE	Returns the number of bytes remaining in the data area.	None	Numeric	System
DATE	Returns a text character string of the current date.	None	Text	System
DOLEVEL	Returns your current level in an external subroutine.	None	Numeric	System
E	Returns the value of natural e (2.71828182845905).	None	Numeric	Mathematical
EXP(a)	Returns the value of natural e to the power of a.	Numeric	Numeric	Mathematical
FALSE	Returns the value zero.	None	Numeric	Boolean
FORMAT	Returns a text-string conversion of a numeric expression according to a supplied edit mask.	Numeric and text string	Text	System
FSI	Indicates the status of a file after an I/O (GET, DELETE, INSERT, or UPDATE).	File-name	Text	File Access
INT(a)	Returns the integer value of a.	Numeric	Numeric	Mathematical
KEY	Returns text character string reflecting the last key pressed in response to a CONVERSE, OBTAIN, PROMPT, or WAIT statement.	None	Text	System
LANGUAGE	Returns the current language function.	None	Text	System



Function	Description	Input	Output	Function
<b>LOG</b> ( <i>a</i> )	Returns the natural logarithm of <i>a</i> .	Numeric	Numeric	Mathematical
<b>LOWERCASE</b> ( <i>t</i> )	Converts a text expression into lowercase.	Text	Text	String
<b>LUID</b>	Returns a text character string of 8 characters containing the VTAM logical unit ID.	None	Text	System
<b>MIXD</b> ( <i>t</i> )	Retrieves DBCS data from mixed-data.	Text	DBCS	String
<b>MIXM</b> ( <i>k</i> )	Returns a mixed-data string containing shift codes from DBCS data.	DBCS	Text	Mixed String
<b>MIXT</b> ( <i>t</i> )	Retrieves text string from mixed-data.	Text	Text	String
<b>MODIFIED</b>	Tests whether a specific field or number of fields within a map definition changed during the last physical I/O.	Map / Field name	Numeric	System
<b>NOT</b> ( <i>a</i> )	Returns TRUE(1) if <i>a</i> evaluates to FALSE(0); otherwise, returns FALSE(0).	Numeric	Numeric	Boolean
<b>NULL</b>	Returns a zero-length text string.	None	Text	String
<b>NUMERIC</b> ( <i>t</i> )	Returns TRUE(1) if the text expression ( <i>t</i> ) contains a valid number; else returns FALSE (0).	Text	Numeric	Boolean
<b>ORD</b> ( <i>t</i> )	Returns the numeric value of the first character's EBCDIC code from <i>t</i> .	Text	Numeric	String
<b>PAD</b> ( <i>statement</i> )	Fills in either or both sides of a DBCS or text variable with a specified character.	Text or DBCS	Text or DBCS	String

Function	Description	Input	Output	Function
PASSWORD	Returns a text character string containing the current password.	None	Text	System
PI	Returns the value of Pi (3.14159265358979).	None	Numeric	Mathematical
POINT( <i>t+t</i> ) ( <i>k+k</i> )	Returns a number representing the position where a string addition or subtraction would occur if you executed it.	Text or DBCS	Numeric	String
PRINTER	Returns a text character string containing the current printer assignment.	None	Text	System
PROGFREE	Returns the number of bytes remaining in the program area.	None	Numeric	System
RELEASE	Returns a text string indicating release information.	None	Text	System
RND( <i>a</i> )	Returns a random real number in the range zero to <i>a</i> , but excluding zero and <i>a</i> .	Numeric	Numeric	Mathematical
SGN( <i>a</i> )	Returns -1 if <i>a</i> < 0, 0 if <i>a</i> = 0, and +1 if <i>a</i> > 0	Numeric	Numeric	Mathematical
SIN( <i>a</i> )	Returns the sine of <i>a</i> where <i>a</i> is in radians.	Numeric	Numeric	Mathematical
SIZE	Returns the size, dimensions, or byte length of a field.	Text or DBCS	Numeric	String

Function	Description	Input	Output	Function
SQLCA	Transfers data between the MANTIS program and the SQL Communications Area.	Varies	Varies	SQL
SQLDA	Allows MANTIS programs to access an SQL Descriptor Area.	Varies	Varies	SQL
SQR( <i>a</i> )	Returns the square root of <i>a</i> .	Numeric	Numeric	Mathematical
TAN( <i>a</i> )	Returns the tangent of <i>a</i> where <i>a</i> is in radians.	Numeric	Numeric	Mathematical
TERMINAL	Returns a text character string of 1–8 characters containing the terminal ID.	None	Text	System
TERMSIZE	Returns terminal size in rows and columns.	None	Text	System
TIME	Returns a text character string of the current system time.	None	Text	System
TRUE	Returns the value +1.	None	Numeric	Boolean
TXT( <i>a</i> )	Returns the text value of <i>a</i> in MANTIS' standard form.	Numeric	Text	String
UNPAD ( <i>statement</i> )	Removes the pad characters from one/both sides of a DBCS or text variable.	Text or DBCS	Text or DBCS	String
UPPERCASE( <i>t</i> )	Converts into uppercase.	Text or DBCS	Text	String

Function	Description	Input	Output	Function
USER	Returns a text string containing the current user name.	None	Text	System
USERWORDS	Returns the number of MANTIS symbolic names currently in use.	None	Numeric	System
VALUE( <i>t</i> )	Returns the numeric value of the text string <i>t</i> .	Text	Numeric	String
VSI	Indicates the highest status within a logical record after a terminal I/O.	File-name	Text	File Access or System
ZERO	Returns the value zero.	None	Numeric	Mathematical

The following table is a reorganization of the functions. The preceding table is organized alphabetically; this table is organized by the type of function: Boolean, File Access, Mathematical, String, or System. Use this table when you want to perform a certain task, but are unsure of the particular function you will need. Scan a category to see other related functions.

## BOOLEAN FUNCTIONS

Function	Description	Input	Output
FALSE	Returns the value zero.	None	Numeric
NOT( <i>a</i> )	Returns TRUE (1) if <i>a</i> evaluates to FALSE (0); otherwise, returns FALSE.	Numeric	Numeric
NUMERIC( <i>t</i> )	Returns TRUE(1) if the text expression ( <i>t</i> ) contains a valid number; else returns FALSE(0).	Text	Numeric
TRUE	Returns the value +1.	None	Numeric

## FILE ACCESS FUNCTIONS

Function	Description	Input	Output
ASI	Indicates the status of a field in a logical view.	Field-name	Text
File variable name	Indicates the status of the last file input or output operation	None	Text
FSI	Indicates the status of file name (VIEW, ACCESS, or FILE).	File-name	Text
VSI	Indicates the status of a RDM view after an I/O (GET, DELETE, INSERT, or UPDATE)	File-name	Text

## MATHEMATICAL FUNCTIONS

Function	Description	Input	Output
ABS( <i>a</i> )	Returns the absolute value of <i>a</i> .	Numeric	Numeric
ATN( <i>a</i> )	Returns the angle in radians whose tangent is <i>a</i> (arctangent).	Numeric	Numeric
COS( <i>a</i> )	Returns the cosine of <i>a</i> where <i>a</i> is in radians.	Numeric	Numeric
E	Returns the value of natural e (2.71828182845905).	None	Numeric
EXP( <i>a</i> )	Returns the value of natural e to the power of <i>a</i> .	Numeric	Numeric
INT( <i>a</i> )	Returns the integer value of <i>a</i> .	Numeric	Numeric
LOG( <i>a</i> )	Returns the natural logarithm of <i>a</i> .	Numeric	Numeric
PI	Returns the value of Pi (3.14159265358979).	None	Numeric
RND( <i>a</i> )	Returns a random real number in the range zero to <i>a</i> , but excluding zero and <i>a</i> .	Numeric	Numeric
SGN( <i>a</i> )	Returns -1 if <i>a</i> < 0, 0 if <i>a</i> = 0, and +1 if <i>a</i> > 0.	Numeric	Numeric
SIN( <i>a</i> )	Returns the sine of <i>a</i> where <i>a</i> is in radians.	Numeric	Numeric
SQR( <i>a</i> )	Returns the square root of <i>a</i> .	Numeric	Numeric
TAN( <i>a</i> )	Returns the tangent of <i>a</i> where <i>a</i> is in radians.	Numeric	Numeric
ZERO	Returns the value zero.	None	Numeric

## SQL FUNCTIONS

Function	Description	Input	Output
SQLCA**	Transfers data between the MANTIS program and the SQL Communications Area.	Varies	Varies
SQLDA**	Allows MANTIS programs to access an SQL Descriptor Area.	Varies	Varies

\*\* Input and output for SQL functions depend on the specific format. See the descriptions in “MANTIS programming language” on page 75 for more information.

## STRING FUNCTIONS

Function	Description	Input	Output
FORMAT	Returns a text-string conversion of a numeric expression according to a supplied edit mask.	Numeric	Text
LOWERCASE( <i>t</i> )	Converts a text expression into lowercase.	Text	Text
MIXD( <i>t</i> )	Retrieves DBCS data from mixed-data.	Text	DBCS
MIXM( <i>t</i> )	Returns a mixed-data string containing shift codes from DBCS data.	(see state-ment descrip-tion) DBCS	Mixed
MIXT( <i>t</i> )	Retrieves text string from mixed-data.	Text	Text
NULL	Returns a zero-length text string.	None	Text
PAD(statement)	Fills in either or both sides of a DBCS or text variable with a specified character.	Text or DBCS	Text or DBCS
POINT( <i>t+t</i> ) ( <i>k+k</i> )	Returns a number representing the position where a string addition or subtraction would occur if you executed it. Normally, this is used with subtraction to indicate the presence of one string within another.	Text or DBCS	Numeric
SIZE	Returns the size, dimensions, or byte length of a field.	Text or DBCS	Numeric
TXT( <i>a</i> )	Returns the text value of <i>a</i> .	Numeric	Text
UNPAD (statement)	Removes the specified characters from one or both sides of a DBCS or text variable.	Text or DBCS	Text or DBCS
UPPERCASE( <i>t</i> )	Converts a text expression into uppercase.	Text	Text
VALUE( <i>t</i> )	Returns the numeric value of the text string <i>t</i> .	Text	Numeric

## SYSTEM FUNCTIONS

Function	Description	Input	Output
ATTRIBUTE	Returns the current status of field, map, terminal, and printer attributes.	Map/ Field-name/ Reserved word	Text
CURSOR	Indicates whether cursor appeared in a specific field at the last terminal I/O.	Field-Name	True/False
DATAFREE	Returns the number of bytes remaining in the data area.	None	Numeric
DATE	Returns a text character string of the current date.	None	Text
DOLEVEL	Returns your current level in an external subroutine.	None	Numeric
LANGUAGE**	Returns the current language function.	None	Text
KEY	Returns text character string reflecting the key you pressed in response to a CONVERSE, OBTAIN, PROMPT, or WAIT statement.	None	Text
LUID***	Returns a text character string of 8 characters containing the VTAM logical unit ID.	None	Text
MODIFIED	Tests whether a specific field, or the number of fields within a map definition, changed during the last physical I/O.	Map/Field-name	Numeric
PASSWORD	Returns a text character string containing the current password.	None	Text
PRINTER	Returns a text character string containing the current printer assignment.	None	Text

\*\* This function is available for service level 5231 and above.

\*\*\* This function is available for service level 5501 and above.



Function	Description	Input	Output
RELEASE	Returns a text string indicating MANTIS release level information.	None	Text
PROGFREE	Returns the number of bytes remaining in the program area.	None	Numeric
TERMINAL	Returns a text character string of 1–8 characters containing the terminal ID.	None	Text
TERMSIZE	Returns terminal size in rows and columns.	None	Text
TIME	Returns a text character string of the current system time.	None	Text
USER	Returns a text character system time string containing the current user name.	None	Text
USERWORDS	Returns the number of MANTIS symbolic names currently in use.	None	Numeric



# 3

## MANTIS programming language

This chapter describes the MANTIS programming statements, commands, and functions in alphabetical order, including the CEF statements. Each description contains the following information (as applicable):

Heading	Description
Restriction	Any overall limitations associated with the command, function, or statement, due to your operating system or installation.
Description	A description of the command, function, statement, parameter, or operand.
Default	The default value, if any, for the operand or parameter.
Format	The required format of the command, function, or statement, followed by a description of all parameters.
Options	The available choices for a parameter such as ON or OFF.
Example	An example of the coded command, function, or statement.
Considerations	Any special limitations, considerations, and guidelines for a specific parameter of the command, function, or statement.
General considerations	Any special limitations, considerations, and guidelines.
Example	An example of the coded command, function, or statement.

## MANTIS language summary

The following table lists and describes the commands, functions, and statements in this chapter by type (some are more than one type). An *(a)* indicates an arithmetic expression. Also included is the mode (either run, immediate, or both for statements), if mode applies to that command, function, or statement. Some entries in the table, such as DATE, are both functions and statements.

Name	Types	Description	Mode
ABS( <i>a</i> )	Function	Returns the absolute value of <i>a</i> .	Run or Immediate
ACCESS	Statement	Identifies external files to be accessed by user program.	Run or Immediate
ASI	Function	Indicates the status of a field in a logical view.	Run or Immediate
ATN( <i>a</i> )	Function	Returns the angle in radians whose tangent is <i>a</i> .	Run or Immediate
ATTRIBUTE	Function, Statement	Alters the attributes associated with a screen design.	Run or Immediate
BIG	Statement	Names and gives dimensions to numeric variables.	Run or Immediate
BREAK	Statement	Exits from FOR-END, UNTIL-END, WHEN-END, and WHILE-END statements.	Run
CALL	Statement	Invokes an interface program.	Run or Immediate
CHAIN	Statement	Replaces the program currently executing with another program and begins executing that program.	Run

Name	Types	Description	Mode
CHR	Function	Returns a text value consisting of the character corresponding to the EBCDIC code specified	Run or Immediate
CLEAR	Statement	Clears the map display, all program data, or data referred to by a symbolic name.	Run or Immediate
COMMIT	Statement	Indicates the completion of a logical unit of work (LUW), or toggles automatic COMMIT.	Run or Immediate
*COMPONENT	Statement	Identifies each component to be assembled by the Compose action or to be disassembled by the Decompose action.	Component Engineering Facility (CEF)
CONVERSE	Statement	Sends a formatted screen design to a terminal and returns the operator's responses to the program.	Run or Immediate
COS( <i>a</i> )	Function	Returns the cosine of <i>a</i> where <i>a</i> is in radians.	Run or Immediate
*CSIOPTNS	Statement	Specifies options to be used at the time of the Compose action.	Component Engineering Facility (CEF)
CURSOR	Function	Indicates whether the cursor appeared in a specific field at the last terminal I/O.	Run or immediate
DATAFREE	Function	Returns the number of bytes available in the data area.	Run or Immediate
DATE ( <i>function</i> ) DATE ( <i>statement</i> )	Function, Statement	Returns the current date in the format specified in your system.	Run or Immediate
DBCS ( <i>statement</i> )	Statement	Names and specifies dimensions for DBCS variables.	Run or Immediate
DELETE	Statement	Deletes a record from a file or a view.	Run or Immediate

Name	Types	Description	Mode
DEQUEUE	Statement	Releases a resource or a TOTAL database record.	Run or Immediate
DO	Statement	Transfers program execution to an internal or external subroutine that returns to the next statement in the original program.	Run
DOLEVEL	Function	Returns your current level in an external subroutine	Run or Immediate
E	Function	Returns the value of natural (e).	Run or Immediate
END	Statement	Physical end of FOR-END, IF-ELSE--END, WHEN-END, WHILE-END, UNTIL-END, or EXEC_SQL-END construct.	Run
ENQUEUE	Statement	Holds a resource.	Run or Immediate
ENTRY-EXIT	Statement	Defines the boundaries of a subroutine or program.	Run
EXEC_SQL-END	Statement	Allows SQL statements to be executed in a MANTIS program.	Run
EXIT	Command, statement	Returns control from a subroutine.	Run or Immediate
EXP(a)	Function	Returns the value of natural (e) to the power of a.	Run or Immediate
FALSE	Function	Returns the value zero.	Run or Immediate
FILE	Statement	Specifies a file that your program accesses.	Run or Immediate
FOR-END	Statement	Repeats execution of a block of statements while a counter is incremented or decremented through a range of values.	Run
FORMAT	Function	Applies an edit mask to a numeric expression.	Run or Immediate

Name	Types	Description	Mode
FSI	Function	Indicates the status of a VIEW or file after an I/O (GET, DELETE, INSERT, or UPDATE).	Run or Immediate
GET	Statement	Reads a record from a file or a view.	Run or Immediate
HEAD	Statement	Centers a heading on the top line of the screen.	Run or Immediate
HELP	Command	Displays explanatory information for an error message, command name, or reserved word.	Immediate
IF-ELSE-END	Statement	Executes a block of statements only if a specified condition is true.	Run
INSERT	Statement	Inserts a new record into a file or view.	Run or Immediate
INT( <i>a</i> )	Function	Returns the integer value of <i>a</i> .	Run or Immediate
INTERFACE	Statement	Specifies an interface that your program accesses.	Run or Immediate
KANJI	Statement	Names and gives dimensions to DBCS variables.	Run or Immediate
KEY	Function	Returns a text string reflecting the key you pressed in response to a CONVERSE, OBTAIN, PROMPT, or WAIT statement.	Run or Immediate
KILL	Command	Terminates a program in a loop. Can be changed or disabled by the Master User.	Immediate
LANGUAGE ( <i>function</i> )	Function, Statement	Returns or assigns a language code to the current task.	Run or Immediate
LANGUAGE ( <i>statement</i> )			

Name	Types	Description	Mode
LET	Statement	Assigns a value to a variable or array elements.	Run or Immediate
LOG( <i>a</i> )	Function	Returns the natural logarithm of <i>a</i>	Run or Immediate
LOWERCASE	Function	Converts a text string into lowercase.	Run or Immediate
LUID	Function	Returns a text string of 8 characters containing the current VTAM logical unit ID.	Run or Immediate
MARK	Statement	Obtains the current position of an RDM logical view established by the last GET, UPDATE, INSERT, or DELETE statement.	Run or Immediate
MIXD	Function	Extracts Kanji from text mixed-data.	Run or Immediate
MIXM	Function	Converts a Kanji expression to a text mixed-data string containing shift codes.	Run or Immediate
MIXMODE	Statement	Controls the handling of mixed-data.	Run or Immediate
MIXT	Function	Extracts a SBCS (single byte character string) text string from text and mixed-data expressions.	Run or Immediate
MODIFIED	Function	Tests whether a field was changed during the last terminal I/O.	Run or Immediate
NEXT	Statement	Executes the next repeat in FOR-END, UNTIL-END, or WHILE-END statements or the next condition in a WHEN-END statement.	Run
NOT( <i>a</i> )	Function	Returns TRUE if <i>a</i> evaluates to FALSE; otherwise, returns FALSE.	Run or Immediate
NULL	Function	Returns a zero-length text value ("").	Run or Immediate



Name	Types	Description	Mode
NUMERIC	Function	Determines if a text expression contains a valid number.	Run or Immediate
OBTAIN	Statement	Obtains data from an unformatted screen and assigns input data to arithmetic and text variables.	Run or Immediate
ORD	Function	Returns the numerical value of the first character EBCDIC code.	Run or Immediate
OUTPUT	Statement	Routes output from CONVERSE or SHOW statements to the screen, printer, or both.	Run or Immediate
PAD	Statement	Fills either or both sides of a text variable with a specified character.	Run or Immediate
PASSWORD	Function	Returns a text string containing the current password.	Run or Immediate
PERFORM	Statement	Invokes a user-written COBOL, Assembler or PL/1 program without passing data to it. Also invokes MANTIS on an external background task in CICS.	Run
PI	Function	Returns the value of PI.	Run or Immediate
POINT	Function	Returns a number representing the position where a string addition or subtraction would occur if you executed it.	Run or Immediate
PRINTER (function) PRINTER (statement)	Function, Statement	Indicates the printer device MANTIS routes output to, or returns the current setting.	Run or Immediate
PROGFREE	Function	Returns the number of bytes available in the program area.	Run or Immediate

Name	Types	Description	Mode
PROGRAM	Statement	Identifies an external subroutine your program uses.	Run or Immediate
PROMPT	Statement	Displays a prompter.	Run or Immediate
RELEASE (function) RELEASE (statement)	Function, Statement	Frees storage for programs loaded with PROGRAM statement, or frees RDM's internal storage for all or individual views currently opened. Returns a text string of current release information.	Run or Immediate
REPLACE	Statement, command	Names the library (yours only), program, password, and description to be created or replaced as the executable program by the Compose action or the program editor.	Run or Immediate
RESET	Statement	Backs out a logical unit of work (LUW).	Run or Immediate
RETURN	Statement	Returns control from a subroutine or stops execution of a program.	Run
RND(a)	Function	Returns a random real number in the range zero to a, but excluding zero and a.	Run or Immediate
RUN	Command	Executes the program currently in the work area.	Immediate
SCREEN	Statement	Specifies a screen design used by your program.	Run or Immediate
SCROLL	Statement	Controls whether MANTIS scrolls top-to-bottom or bottom-to-top on an unformatted screen, or sets scrolling increments for window mode.	Run or Immediate

Name	Types	Description	Mode
SEED	Statement	Seeds the random number generator to generate numbers.	Run or Immediate
SGN( <i>a</i> )	Function	Returns -1 if $a < 0$ , 0 if $a = 0$ , and +1 if $a > 0$ .	Run or Immediate
SHOW	Statement	Displays and formats data on an unformatted screen.	Run or Immediate
SIN( <i>a</i> )	Function	Returns the sine of <i>a</i> where <i>a</i> is in radians.	Run or Immediate
SIZE	Function	Returns the size and dimensions of a field.	Run or Immediate
SLICE	Statement	Limits number of program statements a user can execute before MANTIS suspends the program.	Run or Immediate
SLOT	Statement	Limits the times a program can reach the SLICE limit before MANTIS returns a program loop warning message.	Run or Immediate
SMALL	Statement	Names and gives dimensions to numeric variables.	Run or Immediate
SOURCE	Statement	Names the library (yours only), program, password, and description to be created or replaced as the source program by the Decompose action.	CEF Component Engineering Facility
SQLCA ( <i>function</i> ) SQLCA ( <i>statement</i> )	Function, Statement	Transfers data between the MANTIS program and the SQL Communications Area (SQLCA).	Run or immediate
SQLDA	Function, Statement	Allows MANTIS programs to access an SQL Descriptor Area (SQLDA).	Run or immediate
SQR( <i>a</i> )	Function	Returns the square root of <i>a</i> .	Run or Immediate

Name	Types	Description	Mode
STOP	Statement	Terminates program execution.	Run or Immediate
TAN( <i>a</i> )	Function	Returns the tangent of ( <i>a</i> ) where <i>a</i> is in radians.	Run or Immediate
TERMINAL	Function	Returns a text string of 1–8 characters containing the current terminal ID.	Run or Immediate
TERMSIZE	Function	Returns your terminal size in rows and columns.	Run or Immediate
TEXT	Statement	Names and gives dimensions to text variables.	Run or Immediate
TIME ( <i>function</i> ) TIME ( <i>statement</i> )	Function, Statement	Returns the current time or sets the time format.	Run or Immediate
TOTAL	Statement	Specifies a TOTAL file view.	Run or Immediate
TRAP	Statement	Intercepts error conditions during I/O to a MANTIS, TOTAL, or external VSAM file, or an RDM logical view.	Run or Immediate
TRUE	Function	Returns the value +1.	Run or Immediate
TXT( <i>a</i> )	Function	Returns the text value of <i>a</i> .	Run or Immediate
UNPAD	Statement	Removes zero or more occurrences of a specified character from either or both sides of a text variable.	Run or Immediate
UNTIL-END	Statement	Repeats execution of a block of statements until a specified condition becomes true.	Run
UPDATE	Statement	Replaces a record in a file or view with an updated record.	Run or Immediate
UPPERCASE	Function	Converts a text string into uppercase.	Run or Immediate

Name	Types	Description	Mode
USAGE	Command	Displays program lines containing a specified field name (a Line Editor Command).	Immediate
USER	Function	Returns a text string containing the current user name.	Run or Immediate
USERWORDS	Function	Returns the number of MANTIS symbolic names currently in use.	Run or Immediate
VALUE	Function	Returns the numeric value of a text string.	Run or Immediate
VIEW	Statement	Specifies a RDM logical view.	Run or Immediate
VSI	Function	Indicates the highest field status within a logical record.	Run or Immediate
WAIT	Statement	Temporarily suspends execution of a program.	Run or Immediate
WHEN-END	Statement	Executes a block of statements only when a specified condition is true.	Run
WHILE-END	Statement	Repeats execution of a block of statements while a specified condition is true.	Run
ZERO	Function	Returns the value zero.	Run or Immediate

The rest of this chapter contains all of the MANTIS commands, functions, and statements in alphabetical order.

# ABS

The ABS function returns the absolute value of an arithmetic expression.

**ABS(a)**

**a**

**Description**     *Required.* Specifies the value whose absolute value you want returned.

**Format**            A numeric expression

**General consideration**

See also “SGN” on page 399.

**Examples**        The following examples show how the ABS function returns various absolute values:

Example	Results	Comments
ABS ( 0 )	0	
ABS ( 7 )	7	
ABS ( - 5 )	5	
ABS ( - 13E9 )	.13E11	

---

## ACCESS

The ACCESS statement identifies an external file (VSAM, SAP, or PC) for your program to use. MANTIS retrieves the file view description from the user library and places it in the work area. If view variables are not already defined, MANTIS defines them; if view variables are already defined, MANTIS checks for consistency.

---

```
ACCESS name1([ library1: ]access - name1, password1[, PREFIX] [, n1] [ , NEW
                                                         [, REPLACE ] [, FILE extname])
[, name2([ library2: ]access - name2, password2[, PREFIX][, n2] [ , NEW
                                                         [, REPLACE ] [, FILE extname])... ]
```

---

### *name*

**Description**     *Required.* Specifies a name you use for the file view in subsequent GET, INSERT, DELETE, UPDATE, and TRAP statements.

**Format**            A MANTIS symbolic name (see “[Symbolic names](#)” on page 24)

### **Considerations**

- ◆ When the symbolic name has been previously defined, MANTIS bypasses this definition.
- ◆ The physical open of the file (when required) does not take place until the first GET, UPDATE, INSERT, or DELETE statement is executed. You can use the TRAP statement to trap file open errors.

---

### **[library:]access-name**

**Description**     *Required.* Specifies the name of the file view given during the Design an External File View session.

**Format**            Must be a text expression that evaluates to a valid file view name

**Considerations**

- ◆ This parameter is translated to uppercase upon execution of your program.
- ◆ For more information on valid file view names, refer to *MANTIS Facilities, OS/390, VSE/ESA*, P39-5001.
- ◆ If the file view is in another user's library, you can access it by specifying the name of the user in whose library it does reside, (*library:*). If the file view does reside in your library, you can supply the file view name only. If this parameter is used the colon (:) is required.
- ◆ If you want this entity to be HPO bound, the library name must be specified even if it is in your own library.

---

### ***password***

**Description**     *Required.* Specifies the password valid for the type of access (e.g., read-only, update, insert/delete) your program needs.

**Format**            Must be a text expression which evaluates to the valid password

**Consideration** Password is not translated to uppercase.



---

## PREFIX

**Description**     *Optional.* Specifies whether MANTIS places the symbolic name previously described and an underscore before all data field names associated with this file view. If, for example, you code:

```
10 ACCESS CUSTOMER("CLIENT", "SALES", PREFIX)
```

and the external file CLIENT has a data field named NUMBER, the program would refer to that data field now as CUSTOMER\_NUMBER.

**Format**             Must be coded exactly as shown

### Considerations

- ◆ MANTIS also prefixes the reference variable you supply for SEQUENTIAL and NUMBERED files.
- ◆ See the [PREFIX parameter](#) under “FILE” on page 222.
- ◆ For more information, refer to [MANTIS Program Design and Editing, OS/390, VSE/ESA](#), P39-5013.

---

## *n*

**Description**     *Optional.* Indicates how many buffers MANTIS should allocate to this file.

**Default**             1

**Format**             Arithmetic expression that evaluates to a value in the range 1–255

### Considerations

- ◆ When you use the *n* parameter to specify multiple buffers, you must also add the LEVEL=*n* option to GET, INSERT, DELETE, and UPDATE statements. Variables defined in the ACCESS view then have an additional dimension, that is, scalars become arrays, 1-dimensional arrays become 2-dimensional arrays. So, you cannot use multiple buffers for an external file that has two-dimensional arrays already in it.
- ◆ You can also specify multiple buffers for the reference variable you supply with SEQUENTIAL and NUMBERED files.
- ◆ MANTIS uses only the integer portion of *n*.

---

## NEW

**Description** *Optional.* Marks the data set as reusable by VSAM. This frees any previously inserted records.

**Consideration** This parameter applies to sequential files in Batch mode.

---

## REPLACE

**Description** *Optional.* Marks the data set as reusable by VSAM. This frees any previously inserted records.

**Consideration** This parameter applies to sequential files in Batch mode.

---

## FILE *extname*

**Description** *Optional.* Specifies an external file DDname/DLBL that is used instead of the name provided in the external file view.

**Format** Text expression that evaluates to a valid DDname/DLBL

**Consideration** This parameter applies to sequential files in Batch mode.

### General considerations

- ◆ An external file open (when required) is issued on the first DELETE, GET, INSERT, or UPDATE.
- ◆ Personal computer DIF files are recreated each time you access them and make an insert.
- ◆ File views residing on personal computer files can be accessed only by that personal computer user.
- ◆ For extended external file status messages and Function Status Indicators (FSIs), see “[Extended status messages for MANTIS and external files](#)” on page 521.
- ◆ NEW, REPLACE, and FILE parameters allow you to reuse sequential VSAM files in Batch mode as work files. Sequential files are closed and reopened at a CHAIN without level.

- ◆ This statement can be affected by the External File Exit. See your Master User for details.
- ◆ See also “**DELETE**” on page 183, “**FSI**” on page 232, “**GET**” on page 234, “**INSERT**” on page 277, and “**UPDATE**” on page 479.

PC file type	ACCESS password	Comments	Functions allowed	File pointer*
SEQUENTIAL	VIEW	The file must already exist.	GET	BOF
	ALTER	The file must already exist.	GET	BOF
	INSERT / DELETE	The file will be re-created when the first GET or INSERT is issued.	GET INSERT	BOF EOF
NUMBERED	VIEW	The file must already exist.	GET	BOF
	ALTER	The file must already exist.	GET UPDATE / INSERT / DELETE	EOF
	INSERT / DELETE	The file will be re-created when the first DELETE, GET, INSERT, or UPDATE is issued.	GET UPDATE / INSERT / DELETE	BOF Refer

- \* BOF—Beginning of File  
 EOF—End of File  
 Refer—As determined by the value in the reference variable

## Examples

- ◆ The following example shows how the ACCESS statement is used to retrieve a file and display it in your work area:

```
00010 ENTRY ACCESS_EXAMPLE
00020 .SMALL BUFFER
00030 .ACCESS RECORD("INDEX", "IDXPSWD", 16)
00040 .SCREEN MAP("INDEX")
00050 .WHILE RECORD<>"END" AND MAP<>"CANCEL"
00060 ..CLEAR MAP:BUFFER=1
00070 ..WHILE RECORD<>"END" AND BUFFER<=16
00080 ...GET RECORD LEVEL=BUFFER
00090 ...BUFFER=BUFFER+1
00100 ..END
00110 ..CONVERSE MAP
00120 .END
00130 EXIT
```

- ◆ The following example shows how to access a reusable file for Batch MANTIS:

```
20 .ACCESS RECORD("INDEX", "SERENDIPITY", 16, NEW)
30 .SCREEN MAP("INDEX")
40 .WHILE RECORD<>"END" AND MAP<>"CANCEL"
50 ..CLEAR MAP: LEVEL_NUMBER=1
60 ..GET RECORD LEVEL=LEVEL_NUMBER
100 .END
```

# ASI

The ASI (Attribute Status Indicator) function returns the status of a field in a RDM logical record.

---

**ASI(*view-name*, *field-name*)**

---

## *view-name*

<b>Description</b>	<i>Required.</i> Specifies the name for the logical view.
<b>Format</b>	A MANTIS symbolic name (see “ <a href="#">Symbolic names</a> ” on page 24)

## *field-name*

<b>Description</b>	<i>Required.</i> Specifies the name of a field in a logical view.
--------------------	---

### General considerations

- ◆ Attribute Status Indicator (ASIs) reflect the status of each field defined in your logical view. For details, see “[RDM status functions](#)” on page 517.
- ◆ If the view is defined in a calling program and passed as a parameter on an external DO, both the view and the field-name must be passed.
- ◆ See also “[DELETE](#)” on page 183, “[FSI](#)” on page 232, “[GET](#)” on page 234, “[INSERT](#)” on page 277, “[TRAP](#)” on page 469, “[UPDATE](#)” on page 479, and “[VSI](#)” on page 505.

**Example** The following example shows how the ASI function can test fields:

```
00020 VIEW PARTS ("PARTS-ON-ORDER")
00100 GET PARTS
```

Example	Results	Comments
ASI (PARTS, PART_NAME)	"MISSING"	The field (PART_NAME) is missing; that is, has a null value.

# ATN

The ATN (arctangent) function returns the angle in radians whose tangent is the arithmetic expression (*a*).

**ATN(*a*)**

*a*

**Description**     *Required.* Specifies the value whose arctangent you want returned.

**Format**            A numeric expression

**General considerations**

- ◆ See “COS” on page 164, “PI” on page 367, “SIN” on page 403, and “TAN” on page 454.
- ◆ The value returned will be between  $\pi/2$  and  $-\pi/2$ .

**Examples**            The following examples show how the ATN function returns the value of various numeric expressions:

Example	Results	Comments
ATN( 0 )	0	
ATN( 1 )	.785398163	
ATN( -1 )	-.785398163	

---

# ATTRIBUTE

ATTRIBUTE is both a statement and a function. You can use the statement form to set attributes (to change the current settings) and the function form to return attributes (to inquire about the current settings). Both the ATTRIBUTE statement and function are described in the following pages. The functionality described on these pages is available to all users, but it is controlled by the Master User. See your Master User to be certain you have access to the reserved functions of ATTRIBUTE.

## ATTRIBUTE (Function)

The ATTRIBUTE function returns attributes of a previously-defined SCREEN, field variable, or device, or returns the physical coordinates of the cursor on the screen. Your Master User determines access to the ATTRIBUTE function.




---

MANTIS returns only the nondefault attributes, except for the PROTECTED/UNPROTECTED, AUTOSKIP/NOAUTOSKIP, and UPPERCASE attributes. See the list of attributes at the end of the ATTRIBUTE function.

---



---

ATTRIBUTE	{	( <i>screen - name</i> [ , <i>field - name</i> ] )	}
		(PRINTER)	
		(TERMINAL)	
		(TERMINAL,CURSOR)	

---

---

ATTRIBUTE (screen – name [ ,field - name ]  
[ , (row,col) ] )

---

---

**screen-name**

- Description**  
*Required.* Specifies the name (as defined in a previously executed SCREEN statement) of the screen whose attributes you want returned.
- Format**  
A MANTIS symbolic name (see “Symbolic names” on page 24)

---

**field-name**

- Description**  
*Optional.* Specifies the name of a previously-defined field whose attributes you want returned.
- Format**  
A MANTIS symbolic name (see “Symbolic names” on page 24)
- Considerations**
  - ◆ *field-name* is mutually exclusive with (row,col).
  - ◆ *field-name* is a field in the SCREEN design for *screen-name*.



**(row,col)**

- Restriction** Your Master User determines if you have access to this format of the ATTRIBUTE function
- Description** *Optional.* Specifies the coordinates of the field within the logical display whose attributes you want returned.

**Considerations**

- ◆ The row and column positions must fall within a field in the specified screen domain, or MANTIS issues an error message.
- ◆ MANTIS uses only the integer portion of *row* and *col*.
- ◆ (*row,col*) is mutually exclusive with *field-name*.

**General considerations**

- ◆ The **ATTRIBUTE(screen-name)** function returns one of the following:

- "(lrow,lcol),SET"
- "(lrow,lcol)"

where (*lrow,lcol*) are the row and column domain of the SCREEN. "SET" indicates that *screen-name* is part of the current map set; the absence of "SET" indicates that *screen-name* is not part of the current map set.

- ◆ The **ATTRIBUTE (screen – name  $\left[ \begin{smallmatrix} ,field - name \\ , (row,col) \end{smallmatrix} \right]$ )** function returns:

"(lrow,lcol),length,type,attributes"

where *lrow* and *lcol* are the row and column coordinates of the field in the logical display; *length* is the length of the field, *type* indicates the type of the field, and *attributes* is a list of the 3-character abbreviations of the field's attributes. This function returns only nondefault attributes in uppercase text.

- ◆ This function can return one of the following types for a field. Underlining indicates the 3-character abbreviation that you see when the values are returned.
  - TEXT
  - NUMERIC
  - HEADING
  - KANJI
- ◆ You should use the POINT function to determine if a field has a particular size or attribute. Do not rely on the fixed position of any particular attribute, as the attribute can move when other attributes are set or reset.

---

ATTRIBUTE { (PRINTER) }  
              { (TERMINAL) }

---

---

## PRINTER

**Description**     *Optional.* Specifies that printer attributes are to be returned.

**Format**            A MANTIS reserved word, coded exactly as shown

**TERMINAL**

**Restriction** Your Master User determines if you have access to this format of the ATTRIBUTE function.

**Description** Specifies that printer or terminal attributes are to be returned.

**Format** A MANTIS reserved word, coded exactly as shown

**General considerations**

- ◆ The **ATTRIBUTE**  $\left\{ \begin{array}{l} \text{(PRINTER)} \\ \text{(TERMINAL)} \end{array} \right\}$  function returns text in the following form:

*“(prow,pcol),attributes”*

where *prow* and *pcol* are the size of the physical terminal or printer and *attributes* is a list of the 3-character abbreviations of the terminal's or printer's attributes.

- ◆ You can use the POINT function to determine if that terminal has a particular size or attribute. Do not rely on fixed position of any particular attribute because it can move when other attributes are set or reset (see example 5).

---

**ATTRIBUTE(TERMINAL,CURSOR)**

---

---

**(TERMINAL,CURSOR)**

---

**Description**     *Required.* Returns the position of the cursor in the logical display.

**Format**            MANTIS reserved words, coded exactly as shown

**Considerations**

- ◆ The ATTRIBUTE (TERMINAL,CURSOR) function returns a text string in the following form:

“(lrow,lcol)”

where *lrow* and *lcol* are the row and column positions of cursor within the logical (and not the physical) terminal.

- ◆ If the cursor is in the Command Line or Key Simulation Field, the row is returned as 256.

**General considerations**

- ◆ The ATTRIBUTE function returns the values (nondefault attributes only) of the field, screen, terminal, or printer you are testing. For more information on the attributes, see the first table under “[General considerations for the ATTRIBUTE statement](#)” on page 110, and refer to *MANTIS Facilities, OS/390, VSE/ESA*, P39-5001. For more information on which attributes are returned, see the second table in that section.
- ◆ On the ATTRIBUTE(TERMINAL) function, the NLS value is returned only if it has been set explicitly (has not defaulted.)
- ◆ See also “[CURSOR](#)” on page 171 and “[MODIFIED](#)” on page 332.

**Examples**      The following examples show how the ATTRIBUTE function returns values:

Example	Results	Comments
ATTRIBUTE (MAP, TEST_FIELD)	" ( 9 , 2 ) , 14 , TXT , UPP , UNP , REV , RED , AUT "	TEST_FIELD begins on the screen at logical row 9, column 2, and is 14 bytes long. It is defined as text, uppercase, unprotected, displayed in red reverse video, normal intensity, with autoskip in effect.
ATTRIBUTE (MAP , ( 9 , 2 ) )	" ( 9 , 2 ) , 14 , TXT , UPP , UNP , REV , RED , AUT "	Same field as prior example.
ATTRIBUTE (MAP)	" ( 22 , 80 ) "	The map domain is 22 rows and 80 columns. The map is not part of the current map set.
ATTRIBUTE (PRINTER)	" ( 32 , 80 ) "	Physical 32x80 printer.
ATTRIBUTE (TERMINAL)	" ( 32 , 80 ) , REV , BLI , COL , UNDER , DET "	Physical 32x80 screen that supports reverse video, blinking, color, underlining, and pen detect.
ATTRIBUTE (TERMINAL , CURSOR )	" ( 8 , 5 ) "	Cursor was at logical (within the logical display) row 8, column 5, when user pressed the AID key.
ATTRIBUTE (TERMINAL , CURSOR )	" ( 256 , 1 ) "	Cursor was at unsolicited input field (bottom line) when user pressed the AID key.
POINT (ATTRIBUTE (MAP , TEST_FIELD) - "RED")	TRUE (1)	When TEST_FIELD has "RED" attribute.
( ATTRIBUTE (MAP , TEST_FIELD) - "RED" - "BLU" ) <> ATTRIBUTE (MAP , TEST_FIELD)	TRUE (1)	When TEST_FIELD has either "RED" or "BLU" (blue) attributes.

### ATTRIBUTE (Statement)

Use the ATTRIBUTE statement in your programs to change the attributes of a screen, a field on a screen, a terminal, or a printer. The changed attributes remain in effect until you change them again or use the RESET attribute to revert to the original (default) specification. Use the ATTRIBUTE function at the command line or in a program to check the current status of field-level, map-level, terminal, and printer attributes.

---

ATTRIBUTE	$\left\{ \begin{array}{l} (\text{screen - name} \left[ \begin{array}{l} , \text{field - name} \\ , (\text{lrow}, \text{lcol}) \end{array} \right] ) \\ (\text{PRINTER}) \\ (\text{TERMINAL}) \\ (\text{TERMINAL, CURSOR}) = " (\text{prow}, \text{pcol}) " \end{array} \right\}$	
		= e1,[, e2, e3,...]

---

**ATTRIBUTE screen-name statement**

---

$$\text{ATTRIBUTE } (\text{screen - name} \left[ \begin{array}{l} , \text{field - name} \\ , (\text{lrow}, \text{lcol}) \end{array} \right] ) = e1 [e2, e3, \dots]$$

---

---

**screen-name**

**Description**     *Required.* Specifies the name of a screen design (as defined in a previously executed SCREEN statement).

**Format**             A MANTIS symbolic name (see “[Symbolic names](#)” on page 24)

---

**field-name**

**Description**     *Optional.* Specifies the field whose attribute(s) you want to alter in a screen design (as defined in a previously executed SCREEN statement).

**Format**             A MANTIS symbolic name (see “[Symbolic names](#)” on page 24)

**Consideration**   If you omit *field-name*, the ATTRIBUTE statement applies to all variables associated with the specified screen design (all nonheading fields).

**(Irow,Icol)**

- Restriction** Your Master User determines if you have access to this form of the ATTRIBUTE statement. You get an error message if you try to use this, and you don't have access.
- Description** *Optional.* Represents the initial logical row and column position of the field where you want to assign attributes.
- Format** Two arithmetic expressions that evaluate to a value in the range 1–255

**Considerations**

- ◆ The (Irow, Icol) coordinates for the ATTRIBUTE statement are based on logical coordinates if a CONVERSE statement specifies row, column coordinates other than (1,1) for the screen display. This can be handled in a program by including the screen row and column within the ATTRIBUTE statement. For example:

```
SCRROW=3:SCRCOL=4
CONVERSE MAP(SCRROW,SCRCOL)
ATTRIBUTE(MAP,(5+(SCRROW-1),10+(SCRCOL-1)))="BRI"
CONVERSE MAP(SCRROW,SCRCOL)
```

Sets the bright attribute for a field that was moved from physical display position (5,10) to logical position (7,13) by the CONVERSE of map (3,4).

Following a CLEAR, or a CONVERSE without a WAIT, SET, or UPDATE, for a map not within the current map set, the ATTRIBUTE statement allows the screen definition coordinates to be specified on the ATTRIBUTE statement.

- ◆ If the screen is defined in a caller and passed as a parameter on an external DO, both the screen and the field-name must be passed.
- ◆ Because of the attribute byte, fields normally start at column 2 and beyond.



**e1[,e2,e3...]**

<b>Description</b>	<i>Required.</i> Specifies the attribute(s) for a particular screen, field, or ( <i>lrow,lcol</i> ) designation.
<b>Format</b>	Text expression that evaluates to a comma-separated list of one or more of the attributes available
<b>Options</b>	For available options, see the first table under “ <a href="#">General considerations for the ATTRIBUTE statement</a> ” on page 110. Underlined characters in the following attributes show values that you can use. Do not include spaces (for example, <u>NO</u> COLOR is specified as NOC).

**Considerations**

- ◆ MANTIS translates the expressions to uppercase.
- ◆ If you specify more than one expression, MANTIS processes the expressions from left to right. Within each expression, MANTIS processes the attributes from left to right.
- ◆ MANTIS only looks at the first three characters of an attribute specification, ignoring the rest, up to a comma. Therefore, you may either use the three-character value for brevity or the fully spelled-out attribute for improved program readability. For example, the following two lines are equivalent:

```
ATTRIBUTE (MAP , FIELD) = "CUR "
ATTRIBUTE (MAP , FIELD) = "CURSOR "
```

However, when the first three characters of the spelled-out attribute do not provide the desired attribute value, or the spelled-out attribute includes spaces, you cannot fully spell the attribute out. For example, although the following two lines are valid:

```
ATTRIBUTE (MAP , FIELD) = "NCU "
ATTRIBUTE (MAP , FIELD) = "NCURSOR "
```

...the first of the following two lines is invalid and the second produces an undesired result. In the first line, there is a space. In the second line, “NOC” is not the desired value—“NOC” means NO COLOR rather than “NO CURSOR”.

```
ATTRIBUTE (MAP , FIELD) = "NO CURSOR "
ATTRIBUTE (MAP , FIELD) = "NOCURSOR "
```

**ATTRIBUTE PRINTER/TERMINAL statement**

---

$$\text{ATTRIBUTE } \left\{ \begin{array}{l} \text{(PRINTER)} \\ \text{(TERMINAL)} \end{array} \right\} = e1[, e2, e3...]$$

---

---

**(PRINTER)**

**Description**     *Optional.* Specifies that the online printer is capable of receiving the attributes that follow.

**Format**            A MANTIS reserved word, coded exactly as shown

**Considerations**

- ◆ Use the ATTRIBUTE(PRINTER) statement in environments where printer characteristics vary from device to device, or do not match the characteristics of the attached terminal or where printer exits require certain dimensions or attributes.
- ◆ If the characteristics of a printer are not set, they are assumed to be the same as the attached terminal.
- ◆ *The terminal or printer must support the attributes you specify.* MANTIS does no validation to ensure the device can handle the specified attributes. Device errors or program abends can occur if the device cannot support the generated data stream.

**(TERMINAL)**

**Restriction**      Your Master User determines if you have access to this form of the ATTRIBUTE statement. You receive an error message if you try to use this form and you do not have access.

**Description**      *Optional.* Specifies that the physical terminal is capable of receiving the attributes that follow.

**Format**            A MANTIS reserved word, coded exactly as shown

**Considerations**

- ◆ Use the ATTRIBUTE(TERMINAL) statement in environments where it is impossible to determine the characteristics of a terminal through the assistance of a TP monitor.
- ◆ MANTIS does not support dynamic switching of terminal sizes via the ATTRIBUTE(TERMINAL) statement.
- ◆ If ATTRIBUTE(TERMINAL) is set to UPP, UPPERCASE=N has no effect in the FSE and CEF (Full Screen Editor and Component Engineering Facility).
- ◆ ATTRIBUTE(TERMINAL)=NLS(xxx) can be used to invoke a customizable translation table in countries that use an alternate alphabet. See the description of the NLS attribute, and your System Administrator for more information.
- ◆ *The terminal or printer must support the attributes you specify.* MANTIS does no validation to ensure the device can handle the specified attributes. Device errors or program abends can occur if the device cannot support the generated data stream.

**e1[,e2,e3...]**

<b>Description</b>	<i>Required.</i> Specifies the attribute(s) for a particular screen, field, or (row,col) designation a printer or terminal will accept.
<b>Format</b>	Text expression that evaluates to a comma-separated list of one or more of the attributes available
<b>Options</b>	See the first table under “ <a href="#">General considerations for the ATTRIBUTE statement</a> ” on page 110 for the available options. Underlined characters in the following attributes show abbreviations that you can use. Do not include spaces (for example, <u>NO</u> <u>COLOR</u> is abbreviated NOC).

**Considerations**

- ◆ If you specify more than one expression, MANTIS processes the expressions from left to right. Within each expression, MANTIS processes the attributes from left to right.
- ◆ MANTIS translates the expressions to uppercase.
- ◆ The CLASS attribute for printers allows you to select an operation mode for your printer within the given printer environment. Valid values for printing mode are:
  - CLASS(0)—default printing mode.
  - CLASS(1)—SCS printing support for the 3270 environment.

You also specify the size of the layout sent to the given printer in the format ATTRIBUTE=CLASS (0), (row, col), where the valid values are:

row = 6–255      column = 41–255

For more details on printing modes, refer to [MANTIS Administration, OS/390, VSE/ESA](#), P39-5005.

- ◆ Specify all row/column values relative to (1,1).

---

**ATTRIBUTE TERMINAL/CURSOR statement**


---

**ATTRIBUTE( [TERMINAL,]CURSOR)="(*pro**w*,*pcol*)"**


---

**([TERMINAL,]CURSOR)**

**Description**     *Required.* Specifies that the cursor on the terminal should be positioned at the following row and column coordinates. The TERMINAL reserved word is optional.

**Format**             MANTIS reserved words, coded exactly as shown

---

**"(*pro**w*,*pcol*)"**

**Description**     *Required.* Specifies row and column positions of the cursor at the next CONVERSE.

**Format**             Text expression containing row and column coordinates, separated by a comma and enclosed in parentheses

**Considerations**

- ◆ Specify all row/column values relative to (1,1), as integers separated by a comma and enclosed in parentheses. "*(pro**w*,*pcol*)" can be any text expression. The literal shown here is an example.
- ◆ The CURSOR attribute places the cursor at the first position of the field. If you have multiple ATTRIBUTE statements, MANTIS places the cursor at the first position of the field named in your last ATTRIBUTE statement.
- ◆ The *pro**w* and *pcol* values must be within the dimensions of the physical (and not the logical) terminal.
- ◆ Setting the cursor using this statement has precedence over `ATTRIBUTE(map,field)="CURSOR"`.

General considerations for the ATTRIBUTE statement

The following table lists all the attributes, alphabetically by sets, that are used with the ATTRIBUTE statement to set a value (see “ATTRIBUTE (Function)” starting on page 95 for a list of attributes that are returned by that function). Each attribute is classified by where you can set it—TERMINAL, PRINTER, CURSOR, screen, field, or screen design, as indicated by the headings on the table.

Setting one attribute will disable other attributes on that line. When there are conflicting attributes, MANTIS uses the last attribute that was set. RESET returns attributes to the screen-design values.

You can set some attributes in different places. Generally, you can set any Field attribute at the Screen level, and it then applies to all fields in the screen. (Screen design attributes are discussed further in “ATTRIBUTE (Function)” starting on page 95, and are specified in the Update Field Specifications option of Screen Design as documented in MANTIS Facilities, OS/390, VSE/ESA, P39-5001.)

Attribute	Terminal	Printer	Cursor	Screen	Field	Screen design
AUTOSKIP / NO AUTOSKIP				✓	✓	✓
BLINK / NO BLINK	✓			✓	✓	✓
BOXED / UNBOXED	✓	✓		✓	✓	✓
BRIGHT / NORMAL / HIDDEN	✓			✓	✓	✓
CLASS		✓				
COLOR / NO COLOR	✓	✓				
CURSOR / NO CURSOR				✓	✓	✓

Attribute	Terminal	Printer	Cursor	Screen	Field	Screen design
DEFAULT / RANGE / FILL / MASK / REQUIRED						✓
DETECTABLE / NON DETECTABLE	✓	✓		✓	✓	✓
FULL DISPLAY / NO FULL DISPLAY				✓		✓
HIGHLIGHT/NO HIGHLIGHT				✓	✓	✓
KANJI/NOKANJI	✓	✓				✓
KEEP MAP MODIFIED / RESET MAP MODIFIED				✓		
LEFT BAR / NO LEFT BAR	✓	✓		✓	✓	✓
MIX / NO MIX				✓	✓	✓
MODIFIED / UNMODIFIED				✓	✓	✓
NATIVE LANGUAGE SUPPORT*	✓					
NUMERIC, TEXT, KANJI, HEADING						✓
NEUTRAL / BLUE / PINK / GREEN / TURQUOISE / RED / YELLOW				✓	✓	✓
OVERLINE / NO OVERLINE/	✓	✓		✓	✓	✓
PROTECT BOTTOM LINE / BOTTOM LINE ENTERABLE				✓		✓
PROTECTED / UNPROTECTED				✓	✓	✓
RESET				✓	✓	

Attribute	Terminal	Printer	Cursor	Screen	Field	Screen design
REVERSE VIDEO / VIDEO / REVERSE FULL FIELD	✓	✓		✓	✓	✓
RIGHT BAR / NO RIGHT BAR	✓	✓		✓	✓	✓
SEND ALL FIELDS / SEND MODIFIED FIELDS	✓					
SOUND ALARM / NO ALARM				✓		✓
UNDERLINE / NO UNDERLINE / UNDERLINE FULL FIELD	✓	✓		✓	✓	✓
UPPERCASE / LOWERCASE	✓	✓		✓	✓	✓

\* See the NLS attribute description for the arguments returned with this attribute.



- ◆ All attributes listed in the preceding table might not be available for your particular terminal. MANTIS ignores those attributes that are not supported by your terminal unless changed by ATTRIBUTE (TERMINAL). (Consult your Master User for setting specific terminal or printer attributes.)
- ◆ Use commas to separate multiple attributes in a single text expression (for example, “CUR,HIG”) or multiple expressions (for example, “CUR”, “HIG”).
- ◆ If you specify two or more conflicting attributes (attributes from the same line in the above table), MANTIS uses the last one processed. Related attributes will conflict. For example, of the color attributes, MANTIS will use the last one that was set before the occurrence of a CONVERSE.
- ◆ Data type attributes (“NUM”, “TXT”, and “KAN”) are not allowed to be set by the ATTRIBUTE statement because the SCREEN statement has already defined the associated TEXT, Kanji, BIG, or SMALL variable.
- ◆ You can also set many attributes in Screen Design. (For more details, refer to *MANTIS Facilities, OS/390, VSE/ESA*, P39-5001.)
- ◆ Note that a list of the attributes returned by the ATTRIBUTE function appears in “*ATTRIBUTE (Function)*” starting on page 95.
- ◆ See also “*CURS*OR” on page 171 and “*MODIFIED*” on page 332.

- ◆ The following table, which provides details about the pairs of attributes that can be set by the ATTRIBUTE statement, and those attributes returned by the ATTRIBUTE function, lists attribute abbreviations alphabetically:

Attribute abbreviation	Attribute name	Related attributes	Returned by attribute function	Can be set by attribute statement
ALA	SOUND ALARM	NO ALARM	✓	✓
AUT	AUTOSKIP	NO AUTOSKIP	✓	
BLI	BLINK	NO BLINK/ HIGHLIGHT	✓	✓
BLU	BLUE	NEUTRAL / PINK / GREEN / YELLOW / TURQUOISE / RED	✓	✓
BOT	BOTTOM LINE ENTERABLE	PROTECT BOTTOM LINE	✓	✓
BOX	BOXED	UNBOXED	✓	✓
BRI	BRIGHT	NORMAL / HIDDEN / HIGHLIGHT	✓	✓
CLA	CLASS	None	✓	✓
COL	COLOR	NO COLOR	✓	✓
CUR	CURSOR	NO CURSOR	✓	✓
DEF	DEFAULT VALUE	None	✓	
DET	DETECTABLE	NON DETECTABLE	✓	✓
FIL	FILL	None	✓	
FUL	FULL DISPLAY	NO FULL DISPLAY	✓	
GRE	GREEN	NEUTRAL / BLUE / PINK / TURQUOISE / RED / YELLOW	✓	✓

Attribute abbreviation	Attribute name	Related attributes	Returned by attribute function	Can be set by attribute statement
HED	HEADING	NUMERIC / KANJI / TEXT	✓	
HID	HIDDEN	BRIGHT / NORMAL	✓	✓
HIG	HIGHLIGHT	NO HIGHLIGHT / BLINK / BRIGHT / REVERSE VIDEO	✓	✓
IMX	INTERNAL MIXED	None	✓	
KAN	KANJI	NOKANJI	✓	
KMM	KEEP MAP MODIFIED	RESET MAP MODIFIED	✓	✓
LBA	LEFT BAR	NO LEFT BAR	✓	✓
LOW	LOWERCASE	UPPERCASE		✓
MAS	MASK	None	✓	
MIX	MIX	NO MIX	✓	✓
MOD	MODIFIED	UNMODIFIED	✓	✓
NAU	NO AUTOSKIP	AUTOSKIP		
NCU	NO CURSOR	CURSOR		✓
NEU	NEUTRAL	BLUE / PINK / GREEN / YELLOW / TURQUOISE / RED	✓	✓
NLS	NATIVE LANGUAGE SUPPORT	None	✓	✓
NOA	NO ALARM	SOUND ALARM		✓
NOB	NO BLINK	BLINK		✓
NOC	NO COLOR	COLOR	✓	✓
NOD	NON DETECTABLE	DETECTABLE	✓	✓
NOF	NO FULL DISPLAY	FULL DISPLAY		

Attribute abbreviation	Attribute name	Related attributes	Returned by attribute function	Can be set by attribute statement
NOH	NO HIGHLIGHT	HIGHLIGHT		✓
NOK	NOKANJI	KANJI		
NOL	NO LEFT BAR	LEFT BAR		✓
NOM	NO MIX	MIX		✓
NOO	NO OVERLINE	OVERLINE		✓
NOR	NORMAL	BRIGHT / HIDDEN	✓	✓
NOU	NO UNDERLINE	UNDERLINE / UNDERLINE FULL FIELD		✓
NRB	NO RIGHT BAR	RIGHT BAR		✓
NUM	NUMERIC	HEADING / KANJI / TEXT	✓	
OVE	OVERLINE	NO OVERLINE	✓	✓
PBO	PROTECT BOTTOM LINE	BOTTOM LINE ENTERABLE	✓	
PIN	PINK	NEUTRAL / BLUE / GREEN / YELLOW / TURQUOISE / RED	✓	✓
PRO	PROTECTED	UNPROTECTED	✓	✓
RAN	RANGE CHECK	None	✓	
RBA	RIGHT BAR	NO RIGHT BAR	✓	✓
RED	RED	NEUTRAL / BLUE / YELLOW / GREEN / TURQUOISE / PINK	✓	✓
REQ	REQUIRED	None	✓	
RES	RESET	None		✓
REV	REVERSE VIDEO	VIDEO / REVERSE FULL FIELD	✓	✓

Attribute abbreviation	Attribute name	Related attributes	Returned by attribute function	Can be set by attribute statement
RFF	REVERSE FULL FIELD	REVERSE VIDEO / VIDEO		✓
RMM	RESET MAP MODIFIED	KEEP MAP MODIFIED		✓
SAF	SEND ALL FIELDS	SEND MODIFIED FIELDS	✓	✓
SMF	SEND MODIFIED FIELDS	SEND ALL FIELDS		✓
TUR	TURQUOISE	NEUTRAL / BLUE / PINK / GREEN / RED / YELLOW	✓	✓
TXT	TEXT	HEADING / NUMERIC / KANJI	✓	
UFF	UNDERLINE FULL FIELD	UNDERLINE / NO UNDERLINE		✓
UNB	UNBOXED	BOXED		✓
UND	UNDERLINE	NO UNDERLINE / UNDERLINE FULL FIELD	✓	✓
UNM	UNMODIFIED	MODIFIED		✓
UNP	UNPROTECTED	PROTECTED	✓	✓
UPP	UPPERCASE	LOWERCASE		✓
VID	VIDEO	REVERSE VIDEO / REVERSE FULL FIELD		✓
YEL	YELLOW	NEUTRAL / BLUE / PINK / GREEN / TURQUOISE / RED	✓	✓

## AUTOSKIP

### NO AUTOSKIP

**Description** Specifies whether the cursor will skip automatically to the next unprotected data field when a user fills the current field, or specifies that the terminal can support this attribute.

**Format** AUTOSKIP or NO AUTOSKIP

#### Considerations

- ◆ This attribute can also be specified in Screen Design. Refer to *MANTIS Facilities, OS/390, VSE/ESA*, P39-5001, for more information.
- ◆ This attribute is functionally identical to a following blank-fill character, but uses less storage than a blank-fill character.

---

## BLINK

### NO BLINK

**Description** Specifies whether you want one field, all nonheading fields on the screen to blink, or specifies that the terminal can support this attribute.

**Format** BLINK or NO BLINK

#### Considerations

- ◆ This attribute can also be specified in Screen Design. Refer to *MANTIS Facilities, OS/390, VSE/ESA*, P39-5001, for more information.
- ◆ The three attributes, UNDERLINE, BLINK, and REVERSE VIDEO, are considered as a group to be HIGHLIGHT. If you are specifying these attributes, you only get a choice of one (that is, the attributes are mutually exclusive). When you specify HIGHLIGHT, MANTIS uses the first attribute in the list (UNDERLINE, BLINK, and REVERSE VIDEO) that is supported by the current terminal.
- ◆ See also the “HIGHLIGHT” attribute on page 121.

**BOXED  
UNBOXED**

**Description** Specifies whether a box should be created around one field, all nonheading fields on a screen, or that the terminal can support this attribute.

**Format** BOXED or UNBOXED

**Considerations**

- ◆ This attribute can also be specified in Screen Design. Refer to *MANTIS Facilities, OS/390, VSE/ESA*, P39-5001, for more information.
- ◆ See also the “**HIGHLIGHT**” attribute on page 121.

---

**BRIGHT  
NORMAL  
HIDDEN**

**Description** Specifies whether a field, all nonheading fields on a screen, displays in bright, normal, or hidden intensity. Also specifies that the terminal can support this attribute.

**Format** BRIGHT, NORMAL, or HIDDEN

**Consideration** This attribute can be specified in Screen Design using the INTENSITY attribute. Refer to *MANTIS Facilities, OS/390, VSE/ESA*, P39-5001, for more information.

## **CURSOR**

### **NO CURSOR**

**Description**      Indicates whether the initial cursor position is in the field when you converse the screen.

**Format**            CURSOR or NO CURSOR

#### **Considerations**

- ◆ When more than one field on a screen has the cursor set by screen design, the first one (in a left to right top to bottom order) has the cursor positioned in it. If that first field has the attribute set to NO CURSOR (NCU), the next such field contains the cursor. If there are no more such fields, the hardware positions the cursor in the first unprotected field.
- ◆ Using the statement `ATTRIBUTE(map,field)=CUR` turns off the cursor for all fields in the map except the nominated field. Therefore, when the cursor has been set to multiple fields, the most recent one set will contain the cursor. MANTIS does not remember prior fields' CURSOR settings if you specify NCU for a field that currently has the cursor attribute; MANTIS reverts to the rule in the prior consideration. This statement does not affect any other maps in the map set; any cursor settings for other maps will remain.
- ◆ Using the statement `ATTRIBUTE(map,field)="RES"` or `ATTRIBUTE(map)="RES"` restores the original screen design values for cursor setting. (See the RESET attribute for more information.)
- ◆ This attribute can also be specified in Screen Design. Refer to *MANTIS Facilities, OS/390, VSE/ESA*, P39-5001, for more information.
- ◆ If you specify more than one field containing the cursor, the last specification is used. `ATTRIBUTE(TERMIAL)=(row,col)` takes precedence over all field cursor specifications.
- ◆ `ATTRIBUTE(TERMIAL,CURSOR)=(row,col)` takes precedence over all field cursor specifications.



---

**DETECTABLE**  
**NON DETECTABLE**

- Description** Specifies if one field, or all nonheading fields on a screen will be pen detectable.
- Format** DETECTABLE or NON DETECTABLE
- Consideration** This attribute can also be specified in Screen Design. Refer to *MANTIS Facilities, OS/390, VSE/ESA*, P39-5001, for more information.

---

**FULL DISPLAY**  
**NO FULL DISPLAY**

- Description** Indicates that MANTIS expand the screen size to the dimensions of the current terminal, including the bottom two lines of the screen.
- Format** FULL DISPLAY or NO FULL DISPLAY
- Consideration** If you specify this attribute, you cannot use the key simulation field (or the KILL command). You cannot initiate window mode, and you cannot use the OBTAIN or SHOW statements for CONVERSE.

---

**HIGHLIGHT**  
**NO HIGHLIGHT**

- Description** Indicates that a field or all nonheading fields on a screen are highlighted when they display.
- Format** HIGHLIGHT or NO HIGHLIGHT
- Considerations**
- ◆ This attribute can also be specified in Screen Design. Refer to *MANTIS Facilities, OS/390, VSE/ESA*, P39-5001, for more information.
  - ◆ The three attributes, UNDERLINE, BLINK, and REVERSE VIDEO, are considered as a group to be HIGHLIGHT. If you are specifying these attributes, you only get a choice of one (that is, the attributes are mutually exclusive). When you specify HIGHLIGHT, MANTIS uses the first attribute in the list (UNDERLINE, BLINK, and REVERSE VIDEO) that is supported by the current terminal.

**KEEP MAP MODIFIED**  
**RESET MAP MODIFIED**

**Description** Prevents MANTIS from clearing the modified data tags of a specified screen.

**Format** KEEP MAP MODIFIED or RESET MAP MODIFIED

**Considerations**

- ◆ Specify KMM to prevent MANTIS from clearing modified data tags (or MDT's) of the specified screen or MODIFIED (*map,field*) function. Ordinarily, MANTIS clears MDT's of all maps in the map set for a CONVERSE UPDATE, and for the prior active map in the case of CONVERSE SET. If the MDT's are cleared, a previously modified map returns FALSE for the MODIFIED function. If a map has attribute "KMM", then once modified, the MODIFIED function always returns TRUE until the map is reCONVERSEd or CLEARed. For an explanation of maps and map sets, see the CONVERSE statement.
- ◆ Specify Reset Map Modified (RMM) to turn off KMM and restore ordinary functionality.
- ◆ Use KMM if you will converse additional screens in the map set (e.g., pop-ups) and want to retain the MODIFIED setting for additional validation after the CONVERSE of the additional screen.

---

**LEFT BAR**  
**NO LEFT BAR**

**Description** Indicates that a left bar appears on one field, or on all nonheading fields on a screen.

**Format** LEFT BAR or NO LEFT BAR

**Consideration** This attribute can also be specified using the BOX attribute in Screen Design. Refer to *MANTIS Facilities, OS/390, VSE/ESA*, P39-5001.

---

**MIX**  
**NO MIX**

**Description** Indicates that one field or all nonheading fields on a screen support mixed-data.

**Format** MIX or NO MIX

**Considerations**

- ◆ The field-level attribute MIX enables you to do the codes SO/SI (Shift-out and Shift-in) creation from the terminal.
- ◆ The attribute NO MIX disables the SO/SI creation.
- ◆ This attribute can also be specified in Screen Design. Refer to *MANTIS Facilities, OS/390, VSE/ESA*, P39-5001, for more information.
- ◆ You cannot reset the field-level MIX/NO MIX attribute using the RESET attribute.
- ◆ See “Mixed-data support” on page 579 for more information.

---

**MODIFIED**  
**UNMODIFIED**

**Description** Indicates whether the terminal should send field contents, regardless of whether or not you modified them, to MANTIS when you press **ENTER** or a PF key.

**Format** MODIFIED or UNMODIFIED

**Consideration** The MODIFIED attribute forces the terminal hardware to always send the contents of the field back to the Data Work Area when you press **ENTER** or a PF key. This is normally not required because MANTIS keeps all values sent to the screen. It may be needed in order for some terminal emulators or screen scrapers work.

**NATIVE LANGUAGE SUPPORT**  
**NLS(*xxx*)**

- Description** Specifies an alternate translation table is to be used for upper and lower case translation to and from this terminal, and for the programs running.
- Format** NATIVE LANGUAGE SUPPORT  
that is NLS(*xxx*)
- Options** *xxx* must be null or one of the following:

Code	Language	Code	Language
AFR	Afrikaans	ITS	Swiss Italian
ARA	Arabic	JPN	Japanese
BEL	Byelorussian	KOR	Korean
BGR	Bulgarian	MKD	Macedonian
CAT	Catalan	NLD	Dutch
CHT	Traditional Chinese	NLB	Belgian Dutch
CHS	Simplified Chinese	NON	Norwegian Nynorsk
CSY	Czech	NOR	Norwegian
DAN	Danish	PKL	Polish
GER	German	PTB	Brazilian Portuguese
DES	Swiss German	PTG	Portuguese
ELL	Greek	RMS	Rhaeto-Romanic
ENA	Australian English	ROM	Romanian
ENG	UK English	RUS	Russian
ENU	US English	SKY	Slovakian
ENP	English (uppercase)	SLO	Slovenian

Code	Language	Code	Language
ESP	Spanish	SQI	Albanian
FIN	Finnish	SRB	Serbian (Cyrillic)
FRA	French	SRL	Serbian (Latin)
FRB	Belgian French	SVE	Swedish
FRC	Canadian French	THA	Thai
FRS	Swiss French	TRK	Turkish
GAE	Irish Gaelic	UKR	Ukrainian
HEB	Hebrew	URD	Urdu
HRV	Croatian	U01	User-defined 1
HUN	Hungarian	U02	User-defined 2
ISL	Icelandic	U03	User-defined 3
ITA	Italian	U04	User-defined 4

## Considerations

- ◆ xxx must be one of the languages in the previous list, and must be present in an installation defined table set by your Master User. NLS() resets the value to the current language. The following hierarchy defines this current language. MANTIS defines the language from the information found in these areas:
  - Language set by previously executed LANGUAGE statement.
  - Language defined in the User Profile.
  - Language set by installation default.

See “[LANGUAGE \(Statement\)](#)” on page 306 for more information.

---

## NO COLOR

### NEUTRAL/BLUE/ PINK/GREEN/TURQUOISE/RED/YELLOW

**Description** Indicates that one field or all nonheading fields on a screen display without color or with one of the available colors. Also indicates that the terminal can support this attribute.

**Format** NO COLOR or NEUTRAL/BLUE/  
PINK/GREEN/TURQUOISE/RED/YELLOW

**Consideration** This attribute can also be specified in Screen Design. Refer to *MANTIS Facilities, OS/390, VSE/ESA*, P39-5001, for more information.

---

## OVERLINE

### NO OVERLINE

**Description** Specifies whether a line appears over a field, or over all nonheading fields on a screen.

**Format** OVERLINE or NO OVERLINE

**Consideration** This attribute can also be specified in Screen Design. Refer to *MANTIS Facilities, OS/390, VSE/ESA*, P39-5001, for more information.

---

## PROTECT BOTTOM LINE

### BOTTOM LINE ENTERABLE

**Description** Disallows or allows text to be entered on the bottom line of a screen or map.

**Format** PROTECT BOTTOM LINE or BOTTOM LINE ENTERABLE

#### Considerations

- ◆ This attribute can also be specified in Screen Design. Refer to *MANTIS Facilities, OS/390, VSE/ESA*, P39-5001, for more information.
- ◆ If you specify this attribute, you cannot use the key simulation field (or the KILL command). You cannot initiate window mode, and you cannot use the OBTAIN statement.

**PROTECTED  
UNPROTECTED**

- Description** Disallows or allows one field or all nonheading fields on a screen to be altered.
- Format** PROTECTED or UNPROTECTED
- Consideration** This attribute can also be specified in Screen Design. Refer to *MANTIS Facilities, OS/390, VSE/ESA*, P39-5001, for more information.

---

**RESET**

- Description** Returns attributes to the original specifications made in Screen Design for one field, or all fields and the screen.
- Format** RESET
- Considerations**
- ◆ Using the statement `ATTRIBUTE(map)=RES` also restores the Screen Design values for cursor setting. (See the “*CURSOR/NO CURSOR*” attribute on page 120 for more information.)
  - ◆ The RESET attribute resets all field-level attributes except MIX/NO MIX.

## REVERSE VIDEO

### REVERSE FULL FIELD VIDEO

**Description** Indicates whether a field, screen, or terminal displays in reverse video mode.

**Format** REVERSE VIDEO, REVERSE FULL FIELD, and VIDEO

#### Considerations

- ◆ This attribute can also be specified in Screen Design. Refer to *MANTIS Facilities, OS/390, VSE/ESA*, P39-5001, for more information.
- ◆ The three attributes, UNDERLINE, BLINK, and REVERSE VIDEO, are considered as a group to be HIGHLIGHT. If you are specifying these attributes, you only get a choice of one (that is, the attributes are mutually exclusive). When you specify HIGHLIGHT, MANTIS uses the first attribute in the list (UNDERLINE, BLINK, and REVERSE VIDEO) that is supported by the current terminal.
- ◆ See also the “HIGHLIGHT” attribute on page 121.
- ◆ REVERSE FULL FIELD is supported for compatibility with other MANTIS platforms, and is converted to and reported as the REV option.

---

## RIGHT BAR

### NO RIGHT BAR

**Description** Indicates that a right bar appears on one field, or on all nonheading fields on a screen.

**Format** RIGHT BAR or NO RIGHT BAR

**Consideration** This attribute can also be specified using the BOX attribute in Screen Design. Refer to *MANTIS Facilities, OS/390, VSE/ESA*, P39-5001, for more information.



**SEND ALL FIELDS**  
**SEND MODIFIED FIELDS**

- Description** Indicates whether all fields, or just nonheading (data) fields, are sent to the terminal.
- Format** SEND ALL FIELDS or SEND MODIFIED FIELDS
- Consideration** If a single map is being reCONVERSEd, MANTIS normally only sends the data fields and leaves the heading fields (SMF). When specifying SAF (send all fields), MANTIS will always resend data and heading fields. SAF can be used for terminals and terminal emulators that do not efficiently or correctly handle resending data fields and leaving heading fields.

---

**SOUND ALARM**  
**NO ALARM**

- Description** Specifies whether MANTIS will sound an alarm each time the screen is CONVERSEd.
- Format** SOUND ALARM or NO ALARM
- Consideration** This attribute can also be specified in Screen Design. Refer to *MANTIS Facilities, OS/390, VSE/ESA*, P39-5001, for more information.

## UNDERLINE

## UNDERLINE FULL FIELD

## NO UNDERLINE

**Description** Indicates whether a field or all nonheading fields on a screen are underlined.

**Format** UNDERLINE, UNDERLINE FULL FIELD, or NO UNDERLINE

### Considerations

- ◆ This attribute can also be specified in Screen Design. Refer to *MANTIS Facilities, OS/390, VSE/ESA*, P39-5001, for more information.
- ◆ The three attributes, UNDERLINE, BLINK, and REVERSE VIDEO, are considered as a group to be HIGHLIGHT. If you are specifying these attributes, you only get a choice of one (that is, the attributes are mutually exclusive). When you specify HIGHLIGHT, MANTIS uses the first attribute in the list (UNDERLINE, BLINK, and REVERSE VIDEO) that is supported by the current terminal.
- ◆ UNDERLINE FULL FIELD is supported for compatibility with other MANTIS platforms, and is converted to and reported as the UND option.

## UPPERCASE LOWERCASE

**Description** Indicates whether MANTIS will convert input to uppercase or not (left as entered).

**Format** UPPERCASE or LOWERCASE

### Considerations

- ◆ MANTIS 4.2 screens default to YES for UPPERCASE until you convert them to 5.2 screens and set the default to NO.
- ◆ The translation of UPPERCASE on a systemwide basis follows this hierarchy:
 

C\$OPCUST TRCODE=	Y	Y	Y	N
ATTRIBUTE(TERMINAL)=	LOW	LOW	UPP	Any
Screen design UPPERCASE	N	Y	Any	Any
Translation occurs?	No	Yes	Yes	No
- ◆ This attribute can also be specified in Screen Design. Refer to *MANTIS Facilities, OS/390, VSE/ESA*, P39-5001, for more information.
- ◆ ATTRIBUTE(*map,field*)="UPP" is equivalent to Screen Design UPPERCASE Y.
- ◆ ATTRIBUTE(*map,field*)="LOW" is equivalent to Screen Design UPPERCASE N.

## Examples of the ATTRIBUTE statement

The following example sets the ACCT\_NUM field associated with the screen INVOICE to BRIGHT and PROTECTED:

```
00010 SCREEN INVOICE("PROGRAMA")
00020 ATTRIBUTE(INVOICE,ACCT_NUM)="BRIGHT,PROTECTED"
```

The following example shows several ways to set fields associated with the screen INVOICE to BRIGHT, PROTECTED and CURSOR:

```
00010 SCREEN INVOICE("PROGRAMA")
00020 TEXT STANDARD(16),TEMP(16)
00030 STANDARD="PROTECTED"
00040 TEMP="CURSOR"
00050 ATTRIBUTE(INVOICE,ACCT_NUM)="BRIGHT"      Set to "BRIGHT"
.
.
.
00160 ATTRIBUTE(INVOICE,ACCT_NUM)=STANDARD      Adds "PROTECTED"
.
.
00240 ATTRIBUTE(INVOICE,ACCT_NUM)=TEMP          Adds "CURSOR"
00350 ATTRIBUTE(INVOICE,CUST_NUM)="BRI,PRO,"+TEMP
00360 ATTRIBUTE(INVOICE,CUST_NAME)="BRI,PRO,CUR"

00440 ATTRIBUTE(INVOICE,(3,20))="BRI","PRO","CUR"
00450 ATTRIBUTE(INVOICE,LINEITEM)="BRI",TEMP,STANDARD
```

*One text expression*

*Three text expressions*

The following example resets attributes to screen design specifications:

```
00240 ATTRIBUTE(INVOICE)="RESET"
```

The following example sets all fields in the screen INVOICE to protected and then unprotects only the OK field. This shows how related attributes override, or undo, each other. After the CONVERSE, reset the attributes to those in screen design for the next CONVERSE:

```
02010 ATTRIBUTE(INVOICE)="PRO"
02020 ATTRIBUTE(INVOICE,OK)="UNP"
02030 CONVERSE INVOICE
02040 ATTRIBUTE(INVOICE)="RESET"
```

The following example sets the map-level attributes “PROTECT BOTTOM LINE” and “SOUND ALARM” in the screen INVOICE:

```
03010 ATTRIBUTE(INVOICE)="PBO,ALA"
```

The following example sets the terminal attribute “NATIVE LANGUAGE SUPPORT” to US English (ENU):

```
03010 ATTRIBUTE(TERMINAL) = "NLS(ENU)"
```

The following example sets the terminal attribute “NATIVE LANGUAGE SUPPORT” to the default value.

```
03010 ATTRIBUTE(TERMINAL) = "NLS()"
```

The following example shows how to use the ATTRIBUTE(TERMINAL) and ATTRIBUTE(PRINTER) statements to set values:

```
00010 ATTRIBUTE(TERMINAL) = "COLOR"
```

```
00020 ATTRIBUTE(PRINTER) = "CLASS(1), (66,132)"
```

The following example places the cursor at location (15,15) on the physical terminal (overrides all ATTRIBUTE(*map,field*)=“CUR”):

```
00010 ATTRIBUTE(TERMINAL,CURSOR) = "(15,15)"
```

---

# BIG

The BIG statement names and supplies the dimensions for numeric variables. MANTIS creates an 8-byte numeric floating-point field (or an array of 8-byte fields) and associates it with the name you specify.

---

```
BIG name1[(n1[,n2])]
      [,name2[(n1[,n2])] . . . ]
```

---

---

## *name1*

<b>Description</b>	<i>Required.</i> Specifies the name of the numeric variable.
<b>Format</b>	A MANTIS symbolic name (see “ <a href="#">Symbolic names</a> ” on page 24)
<b>Consideration</b>	When the symbolic name is previously defined, MANTIS bypasses this definition.

---

## *n1*

<b>Description</b>	<i>Optional.</i> Specifies the number of elements in a 1-dimensional array, or the number of rows in a 2-dimensional array.
<b>Default</b>	1
<b>Format</b>	Arithmetic expression that evaluates in the range 1–255
<b>Consideration</b>	MANTIS rounds <i>n</i> to an integer value.

---

***n2***

<b>Description</b>	<i>Optional.</i> Specifies the number of columns in a 2-dimensional array.
<b>Default</b>	1
<b>Format</b>	Arithmetic expression that evaluates to a positive integer in the range 1–255

**Consideration** MANTIS rounds *n* to an integer value.

### General considerations

- ◆ You can specify up to 2048 variable names in a single program.
- ◆ A BIG variable contains a zero upon initial definition.
- ◆ Use BIG (instead of SMALL) to hold numbers involving fractions or more than 6 integer digits. (See “[Numeric data](#)” on page 41.)
- ◆ If *n2* is not specified, or is specified as 1, a one-dimensional array is allocated. If *n1* is not specified, a small scalar variable is allocated.
- ◆ If you do not specify a variable name before its first use, it will default to a BIG data type.
- ◆ See also “[KANJI \(Kanji users only\)](#)” on page 298, “[SMALL](#)” on page 415, “[TEXT](#)” on page 457, and “[Numeric data](#)” on page 41.

### Example

```
00010 X=15
00020 BIG ALPHA(64,3),BETA(12)
```

## BREAK

Use the BREAK statement to exit from a FOR-END, UNTIL-END, WHEN-END, or WHILE-END statement. The statement after the END statement is executed next.

---

### BREAK

---

#### General considerations

- ◆ With nested logic statements, BREAK terminates execution of the innermost FOR-END, UNTIL-END, WHEN-END, or WHILE-END block of statements where it occurs.
- ◆ BREAK from a WHEN-END block of statements bypasses any other WHEN conditions.
- ◆ See also “NEXT” on page 335 and “RETURN” on page 388.

#### Examples

The following example shows how the BREAK statement can be used to exit a FOR-END condition:

```
10 FOR L=1 TO MAXLINES:| For each screen line
20 .GET CUSTOMER LEVEL=L:| Get customer detail fields
30 .IF CUSTOMER="END":| Check status from GET
40 ..BREAK:| Exit FOR Loop if end of file
50 .END to statement 70.
60 END
70 CONVERSE CUST_DETAILS:| Display customer details
```

The following example shows how the BREAK statement can be used to exit a WHEN-END condition:

```
110 WHEN CODE="R"
120 .COLOR="RED"
130 .BREAK <---- will continue at line 210
140 WHEN CODE="B"
150 .COLOR="BLUE"
160 .BREAK <---- will continue at line 210
170 WHEN CODE="G"
180 .COLOR="GREEN"
190 .BREAK <---- will continue at line 210
200 END
210 DO DISPLAY_ORDER
.
.
```



---

## CALL

The CALL statement invokes an interface program. MANTIS calls the program specified in the interface profile and sets the symbolic name variable equal to the status returned by the program.

---

**CALL** *interface*[(*e1,e2...*)]**[LEVEL=*n*]**

---

---

### *interface*

**Description**     *Required.* Specifies the interface name (as defined in a previously executed INTERFACE statement).

**Format**            A MANTIS symbolic name (see “[Symbolic names](#)” on page 24)

---

### *e1, e2*

**Description**     *Optional.* Specifies the elements to be passed to the corresponding element in the interface area (*e1* is the first element in the interface area layout, *e2* is the second, and so on).

**Format**            Must be a valid variable name, or an arithmetic, text, or DBCS expression

**Consideration**   If you don't specify *e1*, ..., *en*, the values passed to the interface program are the current values of the fields defined in the interface layout.

**LEVEL=*n***

<b>Description</b>	<i>Optional.</i> Specifies the buffer number that is passed to the interface program.
<b>Default</b>	1
<b>Format</b>	Arithmetic expression that evaluates to a value in the range of 1 through <i>n</i> , where <i>n</i> is the maximum buffer number, as defined in the corresponding INTERFACE statement

**Considerations**

- ◆ Only specify LEVEL=*n* when the INTERFACE has buffers defined.
- ◆ MANTIS uses only the integer portion of *n*.
- ◆ Variables defined as single level in interface design do not get subscripted by the level number.

**General considerations**

- ◆ Use this statement only after discussing the interfaces that are available to you with your Master User.
- ◆ The interface program is free to do any I/O operations, including I/O operations to the terminal. If you do updates (VSAM, TOTAL, SQL, or RDM), you must issue a COMMIT in the MANTIS program prior to a conversational mode CONVERSE, SHOW, WAIT, or OBTAIN statement. Alternatively, you could specify customization parameter COMFACE=Y which will ensure that a COMMIT is issued at the next terminal I/O. For details about the COMFACE parameter, refer to *MANTIS Administration, OS/390, VSE/ESA*, P39-5005.
- ◆ See TOTAL and VIEW ON/OFF if your interface programs do sign offs or sign ons, or expect a certain state (on or off) when they are invoked.
- ◆ See also “DO” on page 206, “INTERFACE” on page 295, “PERFORM” on page 350, the ON and OFF parameters under “TOTAL” (“TOTAL” starts on page 464), and the ON and OFF parameters under “VIEW” (“VIEW” starts on page 501).

**Example**

The following example shows how the CALL statement is used to invoke an interface called “MASTER(GET)”:

```
00020 INTERFACE MASTER("CUSTOMERS", "ALIBABA", 10)
      .
      .
00060 CALL MASTER("GET", 1234) LEVEL=2
```

---

## CHAIN

The CHAIN statement replaces the program currently executing with another MANTIS program and begins executing that program. MANTIS terminates the issuing program and erases all variables, except those being passed.

---

**CHAIN**"[*library*:] *program-name*"[,*argument1*, *argument2*, . . . ] [ **LEVEL** ]

---

**"**[*library*:] *program-name***"**

**Description**     *Required.* Specifies the name of the MANTIS program you want to load and executes the target program.

**Format**            1–49 character text expression

### Considerations

- ◆ If the program is in another user's library, you can access it by specifying the name of the user in whose library it resides, followed by a colon and the program name: [*library1*:]*program-name*.
- ◆ If the subroutine is in your library, you can specify only the program name.
- ◆ This parameter is translated to uppercase upon execution of your program.
- ◆ If the text expression is enclosed in parentheses, the MANTIS XREF facility will not index the name, even if the name is a text literal inside the parentheses. For example, the following will not cross-reference the MANTIS program "ABC":

```
CHAIN ( "ABC" )
```

## ***argument1, argument2***

**Description**     *Optional.* Specifies the argument(s) to be passed to the new program.

### **Considerations**

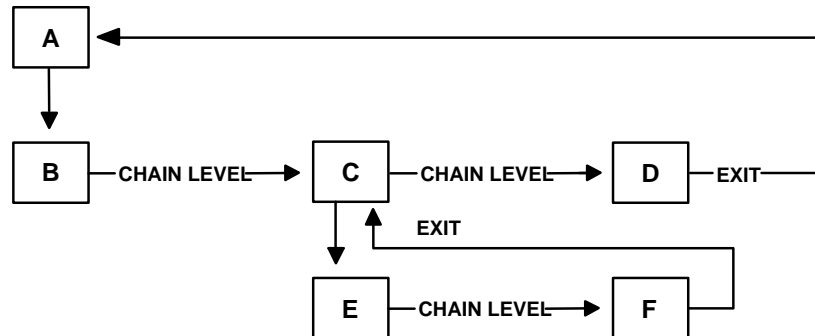
- ◆ The arguments must be previously defined, unsubscripted variable names (constants, literals, reserved words and expressions are not allowed). A maximum of 40 arguments is allowed.
- ◆ The arguments in the CHAIN statement must be simple data types: BIG, SMALL, TEXT, DBCS, or KANJI scalars or arrays. The arguments cannot be symbolic SCREEN, PROGRAM, ENTRY, FILE, INTERFACE, TOTAL, ACCESS, or RDM VIEW names.
- ◆ The arguments in a CHAIN statement define the type and dimensions for the corresponding variables in the program that you are chaining to.
- ◆ All or none of the parameters on the target program's ENTRY statement must be passed as arguments. If no arguments are passed, the target program must define the parameters and assign initial values.

## LEVEL

**Description** *Optional.* Allows an external program to chain to another program that then becomes a subroutine of the original caller, and *not* a new DOLEVEL 0 program. When the CHAINED to program EXITS, it returns to the original calling program.

### Considerations

- ◆ A CHAIN LEVEL at DOLEVEL 0 operates exactly like a CHAIN without the LEVEL.
- ◆ When an external program CHAINs to another program, that program becomes a subroutine of the original caller, not a new DOLEVEL 0 program. If you specify a level, when the CHAINED-to program EXITS, it returns to the original calling program as illustrated below.



- ◆ Arguments passed on a CHAIN ... LEVEL can be defined in the current program, or can have external do parameters that were passed to the current program.

## General considerations

- ◆ If the program is in another user's library, you can access it by specifying the name of the user in whose library it does reside, followed by a colon and the program name:

```
[libraryname:]program-name
```

If the program is in your library, you can specify only the program name.

- ◆ Do not execute this statement while in programming mode without first saving the program that issued the CHAIN. Because the program being chained into the workspace overlays the issuing program, any alterations to the issuing program are lost.
- ◆ While in programming mode, the program being chained to must have the same password as the current program or the current user's password. This restriction does not apply in a program running outside programming mode, for example, if you use the Run a Program by Name facility (refer to *MANTIS Facilities, OS/390, VSE/ESA*, P39-5001).
- ◆ You can pass data from one program to another using the CHAIN statement. This requires an ENTRY-EXIT statement in the target program. The ENTRY-EXIT statement must be the first statement in the program that you are chaining to.
- ◆ This statement can be affected by the Program Load Exit. See your Master User for details.
- ◆ CHAIN without level releases enqueues on external VSAM files created by GET . . . ENQUEUE.
- ◆ The *library:program-name* argument for the CHAIN statement is translated to uppercase upon execution of your program.
- ◆ Since a CHAIN statement leaves a program without a return, MANTIS will not execute any statement following a CHAIN on a program line.
- ◆ Executing a STOP or the main ENTRY's EXIT statement will cause an automatic CHAIN to the user's defined Facility program when run outside programming mode. For details about the facility program specification, refer to *MANTIS Administration, OS/390, VSE/ESA*, P39-5005.
- ◆ See also "DO" on page 206 and "ENTRY-EXIT" on page 213.

## Example

```
00010 ENTRY BUZZ_PHRASE_GENERATOR
00020 .DO SET_UP_VOCABULARY
00030 .HEAD "BUZZ PHRASE GENERATOR"
00040 .CLEAR
00050 .SHOW "I WILL GENERATE A SCREEN FULL OF"
00055 .'" 'BUZZ PHRASES' EVERY"
00060 .'" TIME YOU HIT 'ENTER'. WHEN YOU WANT TO"
00065 .'"STOP, HIT 'PA2'."
00070 .UNTIL KEY="CANCEL"
00080 ..INDEX=1
00090 ..UNTIL INDEX=22
00100 ...A=INT(RND(10)+1)
00110 ...B=INT(RND(10)+1)
00120 ...C=INT(RND(10)+1)
00130 ...SHOW FIRST(A)+" "+SECOND(B)+" "+NOUN(C)
00140 ...INDEX=INDEX+1
00150 ..END
00160 ..WAIT
00170 .END
00180 .CHAIN "GAMES_MENU"
00190 EXIT
```

# CHR

Use the CHR (character) function to return a text value consisting of the character corresponding to the EBCDIC code specified.

## CHR(*a*)

**Description**     *Required.* Specifies the EBCDIC code whose character you want returned.

**Format**            Valid arithmetic expressions in the range of 0–255

**Considerations**

- ◆ MANTIS uses only the integer portion of *a*. For example, CHR(130.5) returns “b”; the .5 is ignored.
- ◆ If the expression's value is outside the valid range, MANTIS will attempt to use the value modulo (MOD) 256; the number is divided by 256 and the remainder is used.

**Examples**

Example	Results	Comments
CHR ( 97 )	" / "	
CHR ( 129 )	" a "	
CHR ( 1024+129 )	" a "	Modulo 256 ignores 1024
CHR ( 193 )	" A "	



The CHR and ORD functions depend on the machine architecture. Results will be different for code moved to an ASCII machine.



## CLEAR

Use the CLEAR statement to clear the scroll map display, clear the data referred to by the symbolic name of a complex entity, or a specific variable name, or clear all program data.

---

```
CLEAR [ name [,...] ]  
      [ ALL ]
```

---

---

### *name*

**Description**     *Optional.* Specifies a symbolic name of a complex entity as defined in an ACCESS, BIG, DBCS, KANJI, FILE, INTERFACE, SCREEN, SMALL, TEXT, TOTAL, or VIEW statement. CLEAR name can also specify a single variable name. MANTIS clears the data referred to by the specified name.

**ALL**

**Description**     *Optional.* Clears the data referred to by all symbolic and variable names in your program.

**General considerations**

- ◆ CLEAR with no parameter:
  - On an unformatted screen, MANTIS clears all data that appeared in the last display of the unformatted screen. Data that was added (via SHOW statements) after the last display and before the CLEAR statement still appears at the next display.
  - On a formatted screen, MANTIS clears the map set. Clearing the map set refers to the list of maps to be displayed on the next CONVERSE without WAIT. It does not refer to resetting the variables associated with the list of maps. To reset all variables associated with all maps in the map set, use individual CLEAR statements with a screen name. For example:

```
CONVERSE MAP1 WAIT
CONVERSE MAP2 WAIT
CLEAR                               : | ( Does not reset any fields in MAP1 or MAP2)
CONVERSE MAP3 SET : | (Displays only fields from MAP3)
```
  - Use clear with no parameter to clear the map set. A new map set is initiated at the next CONVERSE, even if the CONVERSE contains a WAIT or SET.
  - CLEAR without a screen-name does not affect the value of any *screen-name* variable, but sets KEY to "CLEAR".
- ◆ CLEAR with the symbolic name of a numeric variable or array:
  - MANTIS sets the value of the variable (or each element of the array) to zero.
  - If the variables are automatically mapped, the values for related variables are also cleared.

- ◆ CLEAR with the symbolic name of a TEXT or KANJI/DBCS variable or array:
  - MANTIS sets the current length of the variable (or each element of the array) to zero, giving it the null value "".
  - If the variables are automatically mapped, the values for related variables are also cleared.
- ◆ CLEAR with the symbolic name of a SCREEN:
  - MANTIS resets all variables associated with the screen design to zero, if numeric, or a current length of zero, if TEXT or DBCS/KANJI (" or K " ), and sets the value of the *screen-name* variable to empty string(""), but does not affect the KEY function.
  - If variables in the screen are automatically mapped, the values for related variables, fields, and other entities are also cleared.
  - MODIFIED (MAP) and MODIFIED(MAP,FIELD) are FALSE following a CLEAR MAP, until the next CONVERSE MAP.
  - Clearing the screen does not reset the field attributes.
- ◆ CLEAR with the symbolic name of an entity defined by an ACCESS, FILE, INTERFACE, TOTAL, or VIEW statement:
  - MANTIS clears the values of all variables and arrays (including all LEVELs) associated with the entity. In addition, MANTIS clears the value that is returned when the symbolic name is used to invoke a built-in text function. CLEAR resets the values; it does not change the definition of the symbolic names.
- ◆ CLEAR with the ALL parameter:
  - MANTIS clears the values of all the variables and entities defined in the current program. Variables in programs at higher levels are not affected.

## Example

```
10 BIG COUNT, SUBTOTAL(10)
20 FILE CUSTFILE( "CUSTOMERS" , "XANADU" )
30 SCREEN CUSTSCREEN( "CUSTOMER" )
.
.
.
60 CLEAR COUNT, SUBTOTAL, CUSTFILE, CUSTSCREEN
```

The previous example will set the variable COUNT to zero, and each of the 10 elements of SUBTOTAL to zero. The file CUSTFILE will be set to a status of "" and each field in the file view will be set to zero (if numeric) or the current length set to zero (if TEXT or KANJI/DBCS). The last key field pressed in screen CUSTSCREEN will be set to "", and all variables associated with the screen design to zero (if numeric) or a current length of zero (if TEXT or KANJI/DBCS).

## COMMIT

Indicate the completion of a Logical Unit of Work (LUW), or toggle automatic COMMIT processing with the COMMIT statement. COMMIT with no parameters commits pending database updates and prevents them from being backed out by RESET or a system failure. COMMIT with no parameter also flushes updated buffers for the MANTIS cluster and the external files.

---

COMMIT	ON
	OFF

---

---

### ON

<b>Description</b>	<i>Optional.</i> Indicates that you want MANTIS to perform automatic COMMIT processing before each read operation on your terminal (e.g., on every CONVERSE, OBTAIN, PROMPT, SHOW (with a screenful of data) or a WAIT statement).
--------------------	--

---

### OFF

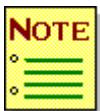
<b>Description</b>	<i>Optional.</i> Indicates that you do not want MANTIS to perform automatic COMMIT processing before each read operation on your terminal.
--------------------	--

### Considerations

- ◆ Automatic COMMIT processing is initially enabled. If the Logical Units of Work in your application are not synchronized, or do not need to be synchronized with terminal reads, use COMMIT OFF to disable automatic COMMIT processing and use the COMMIT statement programmatically to indicate the completion of each Logical Unit of Work.
- ◆ Using the OFF option sets a pseudo-conversational task to a conversational task for the life of the COMMIT OFF. A COMMIT OFF is reset at CHAIN, STOP and fault (run mode), at CHAIN (programming mode); or, until a COMMIT ON is specified.

## General considerations

- ◆ When automatic COMMIT is enabled, (COMMIT set to ON), MANTIS automatically issues a COMMIT prior to any terminal I/O only if an update to the TOTAL DBMS, the RDM logical view, DLI, SQL, or a MANTIS or external file has occurred since the last terminal I/O. You must issue a COMMIT prior to a terminal I/O for file I/O updates executed in an interface program because MANTIS is not aware of these updates. Alternatively, you can set the customization parameter COMFACE=Y which will ensure that a COMMIT is issued at the next terminal I/O. For details on the COMFACE parameter, refer to *MANTIS Administration, OS/390, VSE/ESA*, P39-5005. You can issue COMMITs at other points to control the Logical Unit of Word between terminal I/O.
- ◆ COMMIT with no parameter:
  - MANTIS immediately issues a COMMIT causing all pending updates to be committed.
  - If you issue a COMMIT while holding an enqueued resource, MANTIS automatically dequeues the resource.
  - COMMIT forces a COMMIT on all file types. If you do I/O in an interface, MANTIS commits the I/O.
  - MANTIS commits SQL updates to the SQL database.
- ◆ COMMIT in batch MANTIS programs causes a temporary close on every accessed VSAM file. A high I/O rate can result. Your Master User may choose to disable this function.



---

Issue a COMMIT following a TOTAL statement to avoid locking the system resource table if TOTAL support is in the system and the user profile is set to automatically open the TOTAL database files while executing the TOTAL statement.

---

- ◆ MANTIS for IMS does not support the ENQUEUE, DEQUEUE, and COMMIT statements. To maintain compatibility with other MANTIS versions, these statements may remain in existing programs. MANTIS for IMS will ignore the statements.
- ◆ See also “RESET” on page 387.

**Example**      The following example shows how the COMMIT statement is used within a program with three partial inserts to show that all insertions are complete:

```
000070 WHILE MAINREC<>"END"
000080 ..GET MAINREC
000090 ..INSERT PARTIAL1
000100 ..IF PARTIAL1="ERROR"
000110 ...SHOW"ERROR OCCURRED ON 1ST INSERT":WAIT
000120 ..ELSE
000130 ...INSERT PARTIAL2
000140 ...IF PARTIAL2="ERROR"
000150 ....RESET
000160 ....SHOW"ERROR OCCURRED ON 2ND INSERT, 1ST INSERT BACKED-OUT":WAIT
000170 ...ELSE
000180 ....INSERT PARTIAL3
000190 ....IF PARTIAL3="ERROR"
000200 .....RESET
000210 .....SHOW"ERROR OCCURRED ON 3RD INSERT, 1ST & 2ND INSERT BACKED-OUT":WAIT
000230 ....ELSE
000240 .....COMMIT
000250 .....SHOW"ALL THREE INSERTS SUCCESSFUL, PROCESSING NEXT RECORD":WAIT
000260 ....END
000270 ...END
000280 ..END
000290 .END
```

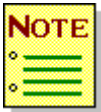
The following example uses COMMIT OFF:

```
10  ENTRY  CUSTOMER_PROG
20  SCREEN CUST_SCR( "CUSTOMER_INPUT_SCR" )
30  ACCESS CUST_FILE( "CUSTOMER_FILE", "XRPT" )
40  ACCESS CUST_CHECK( "CUSTOMER_CHECK", "XRPT" )
50  COMMIT OFF                                Task is set to conversational
60  WHILE CUST_FILE < > "END"
70  .CONVERSE CUST_SCR                        No automatic COMMIT occurs
80  .IF CUST_FILE_NUMBER = CUST_CHECK_NUMBER
90  ..UPDATE CUST_FILE
100 .COMMIT                                  Updates are committed,
                                           end of LUW
110 .END
.
.
.
150 .END
```



# COMPONENT

When coded in a MANTIS source program, the COMPONENT statement identifies each component that can be assembled by the Compose action into expanded component code in a composed (executable) program. When displayed in a MANTIS composed executable program, the COMPONENT statement identifies each component that can be nominated and disassembled by the Decompose action into separate, updated components.



If UPPERCASE = N has been specified in the Full Screen Editor, and you are editing a composed (executable) program, the term COMPONENT appears behind a comment bar, and therefore is not translated into uppercase by MANTIS. If you nominate this component and then attempt a Decompose, it will fail because the term COMPONENT is not recognized in lowercase format. If your terminal is set to UPPERCASE = N, make sure you enter the term COMPONENT in uppercase from the keyboard.

**COMPONENT**"[*library:*] *component-name* [/password][/*description*]"

**library:**

<b>Description</b>	<i>Optional.</i> Specifies the name of the user's library where the component resides.
<b>Default</b>	Your sign-on library name
<b>Format</b>	A MANTIS symbolic name, 1–16 characters in length, followed by a colon (:) (see “ <a href="#">Symbolic names</a> ” on page 24)
<b>Consideration</b>	Although you can only compose a source program that resides in your user library, you can specify a component from another user's library in the COMPONENT statement.

**component-name**

<b>Description</b>	<i>Required.</i> Specifies the name of the component that is expanded in the executable program.
<b>Format</b>	A MANTIS symbolic name, 1–32 characters in length (see “ <a href="#">Symbolic names</a> ” on page 24)

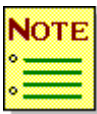
**/password**

<b>Description</b>	<i>Conditional.</i> Specifies the password used to previously save the component.
<b>Default</b>	Your sign-on password or last program password specified, preceded by a slash (/)
<b>Format</b>	A MANTIS symbolic name, 1–16 characters in length (see “ <a href="#">Symbolic names</a> ” on page 24)
<b>Consideration</b>	Required only if the component resides in another user’s library and the program password is different than your sign-on password.

---

**/description**

<b>Description</b>	<i>Optional.</i> Specifies the text description of the component.
<b>Format</b>	1–46 characters of text



---

The description on the COMPONENT statement is optional and is used only for user reference purposes. It is not used to create or update the description when the component is saved or replaced.

---

**General considerations**

- ◆ The COMPONENT statement is *required* when using Component Engineering Facility (CEF). At least *one* COMPONENT statement is required in the source program for the Compose action to work, or an error message is displayed when you attempt to issue the Compose action.
- ◆ Code one COMPONENT statement in a MANTIS *source* program for each component you want to use in your application design. If a component contains an ENTRY/EXIT statement, code the COMPONENT statement(s) after the last EXIT statement because nested ENTRY/EXIT statements are not permitted. This is true only if your components are bounded by ENTRY/EXIT statements (that is, used for INTERNAL DOs). Components can be used to define large blocks of global variables where an internal DO is not desirable. Other SQL uses include creating INSERT and SELECT lists as well as using COMPONENT in any other SQL statement.

- ◆ Double quotes ("" ) (or another character in some installations) are required around the parameters of the COMPONENT statement, as shown in Example 4. The colon (:) is required to separate library and program name, and the slash character (/) is required to separate password and description as shown.
- ◆ A text expression cannot be used in a component statement, the library, component-name, password, and description must be supplied as literals within quotes.
- ◆ COMPONENT statements cannot be continued from one line to the next. Be sure each COMPONENT statement is a single statement coded on a single line of a MANTIS source program.
- ◆ You can select the COMPONENT statement in the Full Screen Editor with the S (select) line command for nested editing. For more information about the S line command, refer to *MANTIS Program Design and Editing, OS/390, VSE/ESA*, P39-5013.
- ◆ Components named in the COMPONENT statement(s) result in expanded component code in the composed (executable) program when you issue the Compose action on the source program. Components are expanded in the order of the COMPONENT statements coded in the source program.
- ◆ Source programs are not executable.
- ◆ See also “CSIOPTNS” on page 165, “DO” on page 206, “ENTRY-EXIT” on page 213, “REPLACE” on page 383, and “SOURCE” on page 417.
- ◆ For more information about the use of the COMPONENT statement and the Compose and Decompose actions, refer to *MANTIS Program Design and Editing, OS/390, VSE/ESA*, P39-5013.

**Examples** The following examples show the COMPONENT statement as it appears in a source program, a composed (executable) program, and nominated in a composed (executable) program for the Decompose action. To nominate a COMPONENT statement, change the asterisk (\*) to an at sign (@) as shown:

Example	Comments
00010 COMPONENT"ACCT:CUST_ERROR_PROC"	Before Compose.
00010  *COMPONENT"ACCT:CUST_ERROR_PROC"	Composed (executable) program after Compose.
00010  @COMPONENT"ACCT:CUST_ERROR_PROC"	Composed (executable) program, component nominated for update in Decompose.

◆ The following example shows how to use the COMPONENT statement to add several different components to your program:

```
00010 ENTRY CUST_INSERT
00020 REPLACE"ACCT:CUST_INSERT/DEPT1234/CUSTOMER RECORD INSERT PROGRAM"
00030 CSIOPTNS"COMMENTS=YES:FORCE=NO:SEQUENCE 10,10"
.
.
.
00520 EXIT
00530 COMPONENT"ACCT:CUS_INIT_FILE_HEADER/DEPT1234/INITIALIZE HEADING"
00540 COMPONENT"ACCT:CUS_ERROR_PROC/DEPT1234/ERROR PROCESSING ROUTINE"
00550 COMPONENT"ACCT:CUS_TERMINATE/DEPT1234/TERMINATE ROUTINE"
```



\* A character other than @ may be used at your installation. See your System Administrator if you have trouble using the @ character.

---

## CONVERSE

The CONVERSE statement sends a formatted screen design or map set to a terminal and returns any response or alterations to the program.

---

```
CONVERSE screen - name [(row1,col1)] [WAIT  
SET  
UPDATE] [WINDOW  
DISPLAY] [(row2,col2)]  
RELEASE
```

---

### *screen-name*

<b>Description</b>	<i>Required.</i> Specifies the name of an existing screen design, (as defined in a previously executed SCREEN statement).
<b>Format</b>	A MANTIS symbolic name (see “ <a href="#">Symbolic names</a> ” on page 24)

---

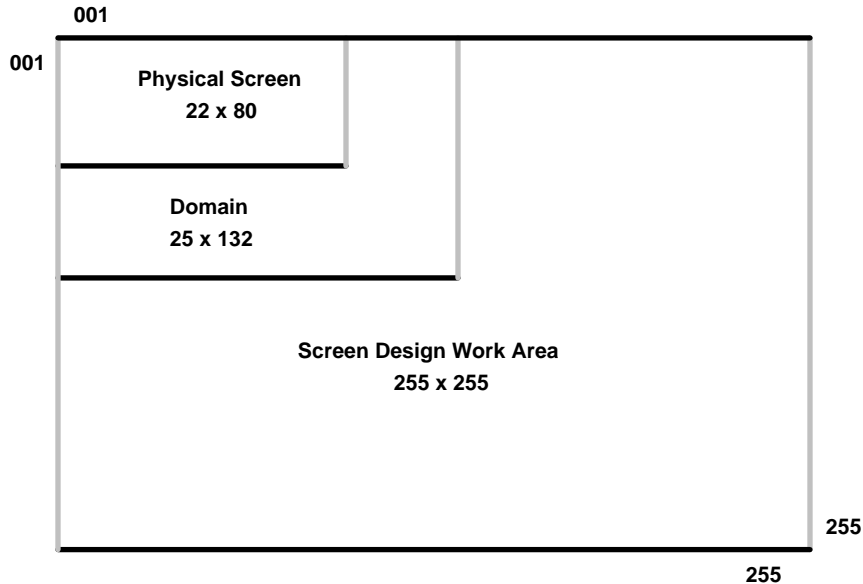
### *row1,col1*

<b>Description</b>	<i>Optional.</i> Specifies the dynamic offset (row and column positions) of a screen within the logical display.
<b>Format</b>	Two arithmetic expressions that evaluate from 1 to 255
<b>Default</b>	(1,1)

### Considerations

- ◆ All row/column values that you supply must be relative to row 1, column 1. Enter row 2, column 3 as (2,3).
- ◆ MANTIS uses only the integer portions of row1 and col1.
- ◆ MANTIS accepts row and column values only if the domain of the screen does not extend beyond or across the logical display boundaries (255 by 255).

The figure below illustrates the relationship between the physical screen, domain, and the work area:



---

**WAIT**

**Description**     *Optional.* Tells MANTIS to add this screen to a map set without displaying the map set on the terminal. This screen is passive in the map set (input fields are protected) unless the UPDATE option is used on a subsequent member of the map set.

---

**SET**

**Description**     *Optional.* Tells MANTIS to add this screen to a map set and to display the entire set on the terminal. This is the active map in the map set. (Active map fields appear on top, that is, overlapping fields from active maps have precedence.)

---

**UPDATE**

**Description**     *Optional.* Tells MANTIS to add this screen to a map set and to display the map set. This map is active within the map set. Fields on passive map(s) that are unprotected and completely displayed can also be altered.

---

**WINDOW**

**Description**     *Optional.* Enables window mode with the window positioned at (row2,col2). (For instructions on using window mode, refer to *MANTIS Facilities, OS/390, VSE/ESA*, P39-5001.)

---

**DISPLAY**

**Description**     *Optional.* Specifies that the physical display be positioned at row2, col2. Window mode is not automatically activated.

---

**(row2,col2)**

**Description**     *Optional.* Used with the WINDOW or DISPLAY options. Indicates the row/column position for the physical display.

**Format**            Two arithmetic expressions that evaluate from 1 to 255

**Default**            (1,1)

**Considerations**

- ◆ All row/column values that you supply must be relative to row 1, column 1. Enter row 2, column 3 as (2,3).
- ◆ MANTIS uses only the integer portions of row1 and col1.

## RELEASE

**Description**     *Optional.* Removes the specified map from the map set.

**Consideration** If a CONVERSE *map* RELEASE is issued for a specified map that is not in the map set, MANTIS issues an error message.

### General considerations

- ◆ When you issue CONVERSE (other than WAIT or RELEASE options), MANTIS:
  - Displays the screen design with all edited data on the terminal and, unless FULL DISPLAY is indicated, fills the Message Line with any unfinished SHOW statements (terminated by ;).
  - Waits until you press a program-interrupt key (such as ENTER, PF key, PA key, or CLEAR).
  - Checks that the input numeric fields contain valid numeric characters and that they correspond to the field mask characters.
  - Saves any alterations you make into the corresponding program variables.
  - Saves a text value (associated with the screen name) describing the terminal key that you pressed to cause the input (or what you entered in the Key Simulation field).
  - Saves any unsolicited data you enter in the bottom line of the screen to feed the next OBTAIN statement encountered.
  - Redisplays the same screen with an error message on the second-to-last line if you enter any invalid numeric fields, and highlights the faulty fields on the screen (without losing the numeric or hidden attribute).
  - Resets the SLOT and SLICE counters to zero if the CONVERSE is to a display terminal. If the CONVERSE is to a printer, then the SLOT and SLICE counters are not reset.



- ◆ Because MANTIS reserves the last two lines on all screens (unless you specify the FULL DISPLAY attribute through the Screen Design facility or with an ATTRIBUTE statement), they appear to be blank. The last line has two fields—Command Line and Key Simulation. You can enter data on the last line by positioning the cursor:
  - In the left field to enter data (see the OBTAIN statement).
  - In the right field to simulate a program function key (e.g., if you enter PF7, MANTIS assumes that you pressed the PF7 key). You can also use the right field on the last line to terminate a program (e.g., enter KILL) or enter window mode.
- ◆ On input numeric fields with decimal fractions, you need only enter the significant decimal digits. The input numeric field cannot contain more digits than the mask allows.
- ◆ A CONVERSE (causing physical terminal I/O) automatically dequeues resources if the MANTIS user is defined as pseudoconversational in CICS or COMMITs resources in CICS.
- ◆ A CONVERSE statement that contains the WAIT, SET, or UPDATE option adds the map to the current map set unless it is the first converse or is preceded by a CLEAR statement; for example:

CONVERSE MAP1	<i>(Sent to terminal)</i>
CLEAR	<i>(Initiates new map set and clears screen)</i>
CONVERSE MAP2 WAIT	<i>(MAP2 added to empty map set)</i>
CONVERSE MAP3 SET	<i>(Screen containing MAP2, MAP3 sent to terminal)</i>

- ◆ The current map set is cleared when a CLEAR statement (with no screen name) is issued, or when a CONVERSE statement has neither the WAIT, SET or UPDATE options specified, for example:

CONVERSE MAP1	<i>(Sent to terminal)</i>
CONVERSE MAP2 SET	<i>(MAP2 added to map set containing MAP1 and sent to terminal)</i>
CONVERSE MAP3	<i>(Map set cleared, and MAP3 is sent to terminal)</i>

- ◆ A map added to a map set already containing that map results in a change of ordering within the map set. If you specify SET or UPDATE, the map moves from passive to active.
- ◆ A map can appear only once within a map set. If it is added twice (with a WAIT, SET, or UPDATE option), it is moved to the new dynamic offset. If you want the same screen image to appear twice, you must have two screen variables and converse both of them, for example:

```
SCREEN MAP1 ( " INDEX " ) ,MAP2 ( " INDEX " )  
CONVERSE MAP1  
CONVERSE MAP2 ( 20 , 30 ) SET
```

- ◆ Overlapping fields have precedence of order when they are conversed. Later maps' fields have priority over earlier maps' fields.
- ◆ Fields that are partially displayed because they overlap the physical screen boundary can be updated.
- ◆ If you create a screen on a large terminal and converse the screen on a smaller terminal, MANTIS does not automatically put you in window mode. The program must use the WINDOW option or the user must enter W in the Key Simulation Field to get into window mode, then use the PF keys to scroll around the screen.
- ◆ This statement can be affected by the Printer Write and Terminal Write Exits. See your Master User for details.
- ◆ See also "KEY" on page 301, "KILL" on page 303, "OBTAIN" on page 341, "SCREEN" on page 393, and "SHOW" on page 400.

## Examples

- ◆ The following example shows how to use the CONVERSE statement with WAIT and SET to add one map without displaying it, and then add a second map and display both maps:

```

00010 ENTRY INDEX
00020 .FILE RECORD("INDEX","SERENDIPITY")
00030 .SCREEN MAP1("INDEX")
00035 .SCREEN MAP2("DETAILS")
00040 .GET RECORD FIRST
00050 .WHILE RECORD<>"END" AND MAP<>"CANCEL"
00055 ..CONVERSE MAP2(5,10)WAIT
00060 ..CONVERSE MAP1 SET
00070 ..WHEN MAP1="PF1"
00080 ...INSERT RECORD
00090 ..WHEN MAP1="PF2"
00100 ...UPDATE RECORD
00110 ..END
00120 ..GET RECORD
00130 .END
00140 .STOP
00150 EXIT

```

- ◆ The following example shows how to use the CONVERSE statement with WAIT, SET, UPDATE, and RELEASE. Maps can be added, displayed, and then removed in any order. This example shows map 2 being RELEASEd before map 3.

```

00010 SCREEN MAP1(ALPHA),MAP2(BETA),MAP3(GAMMA)
00050 CONVERSE MAP1 WAIT
00060 CONVERSE MAP2 SET
00070 CONVERSE MAP3 UPDATE
00080 CONVERSE MAP2 RELEASE
00090 CONVERSE MAP3 UPDATE

```

# COS

The COS function returns the cosine of *a* where *a* is in radians.

**COS(*a*)**

*a*

**Description**     *Required.* Specifies the value whose cosine you want returned.

**Format**            An arithmetic expression

**Consideration** If the angle is in degrees, it must be converted to radians. See “PI” on page 367 to find out how to do this.

**General consideration**

See also “ATN” on page 94, “PI” on page 367, “SIN” on page 403, and “TAN” on page 454.

**Examples**            The following examples show how the COS function is used to return the cosine:

Example	Results	Comments
COS(0)	1	
COS(PI)	-1	
COS(10)	-.839071529	

---

## CSIOPTNS

The CSIOPTNS statement specifies the values of three options used to execute the Compose action.




---

If UPPERCASE=N has been specified in FSE (Full Screen Editor), you must enter the entire CSIOPTNS statement in UPPERCASE mode for it to be recognized by MANTIS.

---



---

**CSIOPTNS" [COMMENTS =  $\begin{Bmatrix} \text{YES} \\ \text{NO} \end{Bmatrix}$ ] [: FORCE =  $\begin{Bmatrix} \text{YES} \\ \text{NO} \end{Bmatrix}$ ] [: SEQUENCE[*m*, *n*]]"**

---

**COMMENTS=**

<b>Description</b>	<i>Optional.</i> Specifies whether the composed (executable) program that results from the Compose action contains commented <code> *COMPONENT</code> statements (to indicate the beginning of components) and <code> *CEND</code> statements (to mark the end of components).
<b>Default</b>	YES
<b>Format</b>	Text literal YES or NO
<b>Options</b>	<p>YES Comments COMPONENT statements in executable programs.</p> <p>NO Omits COMPONENT and CEND statements from executable programs.</p>

**Considerations**

- ◆ If the COMMENTS parameter is coded COMMENTS=YES in the CSIOPTNS statement in the source program, the composed (executable) program contains commented COMPONENT statements that begin with the vertical bar ( | ), for example `|*COMPONENT`. In addition, COMMENTS=YES also supplies the commented `|*CEND` statement to mark the end of an individual component in the executable program.
- ◆ If the COMMENTS parameter is coded COMMENTS=NO, the `|*COMPONENT` and `|*CEND` statements are not included in the composed (executable) program to identify individual components. You can use this when these comments would interfere; for example, in the middle of an EXEC\_SQL-END construct.



---

For the Decompose action to work on a composed (executable) program, the COMPONENT statements and CEND statements must be present in the composed (executable) program.

---

- ◆ If the COMMENTS parameter is omitted or misspelled, the system default value is used.
- ◆ You can override the COMMENTS parameter from the COMPOSE Program Entry screen. This screen displays the Function Option *Component stmt?*. For information about this screen and option and the valid values, refer to *MANTIS Program Design and Editing, OS/390, VSE/ESA*, P39-5013.

---

**FORCE=**

<b>Description</b>	<i>Optional.</i> Specifies whether the Compose action is forced to occur if a composed (executable) program changed since the last time the Compose action was issued on the source program.
<b>Default</b>	NO
<b>Format</b>	Text literal YES or NO
<b>Options</b>	YES To force the Compose action. NO To bypass the Compose action.

**Considerations**

- ◆ If the FORCE parameter is coded FORCE=NO in the CSIOPTNS statement in the source program, the Compose Confirmation screen is displayed (if you changed a composed (executable) program and then attempt to issue the Compose action on its source version). If the FORCE parameter is coded FORCE=YES, the Compose Confirmation screen is not displayed and the changes you made to the composed (executable) program is overlaid by the Compose action.
- ◆ If the FORCE parameter is omitted or misspelled, the system default value is used.
- ◆ You can override the FORCE parameter if you issue the Compose action from the COMPOSE Program Entry screen. This screen displays the Function Option *Force compose?* to allow you to either accept or override the values of the FORCE parameter. For information about this screen and option, refer to *MANTIS Program Design and Editing, OS/390, VSE/ESA*, P39-5013.
- ◆ NO (the default) is the normal condition of FORCE=. YES could be used when composing an entire application in Batch MANTIS.

**SEQUENCE *m,n***

<b>Description</b>	<i>Optional.</i> Specifies how the line numbers of the composed program are sequenced before the program is replaced.
<b>Default</b>	10,10
<b>Format</b>	<i>m,n</i> are 1–3 digit numbers separated by a comma (expressions are not allowed)
<b>Options</b>	Enter two digits (e.g., 5,5) separated by a comma. The first digit indicates the starting line number; the second digit indicates how much each succeeding line number is incremented.

**Considerations**

- ◆ With the exception of the FSE, the SEQUENCE parameter is the only location where you can alter the sequence number of program lines in a composed (executable) program.
- ◆ The set of numbers is optional, but if you code one, you must code the other. If one of the digits (or the comma) is omitted, the default value SEQUENCE 10,10 is used. If the SEQUENCE parameter is omitted or misspelled, the system default value is used.
- ◆ If the SEQUENCE parameter is coded SEQUENCE 1,1 the line numbers in the composed (executable) program appear as 1, 2, 3, and so forth.
- ◆ Coding SEQUENCE 0,0 calculates an optimum sequence number for the program lines.
- ◆ If the syntax of the SEQUENCE parameter is incorrect, or the maximum line number is exceeded, you will receive an error message.
- ◆ The largest line number in MANTIS is 30000, and the SEQUENCE parameter must be set within this limit.



## General considerations

- ◆ The CSIOPTNS statement is optional. If used, it cannot be continued from one line to the next. The CSIOPTNS statement must be coded as a single statement on a single line in a MANTIS source program. The recommendation shown in the previous example is to code the CSIOPTNS statement after the ENTRY and REPLACE statements in your program for consistency and readability.
- ◆ The examples in this manual show the parameters coded in the CSIOPTNS statement in the order of COMMENTS, FORCE, SEQUENCE. However, you can code these parameters in any order. Each parameter is optional so you can code any or all of them.
- ◆ Double quotes (") (or another user-defined character for some installations) are required around the parameters of the CSIOPTNS statement as shown in the example below. The colon (:) is required to separate the parameters, and the equal sign (=) is required between the parameter and the value of COMMENTS and FORCE. There must be a space between SEQUENCE and the value as shown below.
- ◆ Because options are being set for a whole program at one time, you can only use comments one at a time. You cannot toggle comments ON and OFF throughout the program.
- ◆ See also "**COMPONENT**" on page 153, "**REPLACE**" on page 383, or "**SOURCE**" on page 417.

Examples

- ◆ The following examples show the CSIOPTNS statement as it appears in a source program and a composed (executable) program

Example	Comments
00010 CSIOPTNS"COMMENTS=NO"	Before Compose (Source program).
00010  *CSIOPTNS"COMMENTS=NO"	Composed (executable) program after Compose.

- ◆ The following example shows how the CSIOPTNS is used to set FORCE to “YES”, COMMENTS to “NO”, and SEQUENCE to “5,5”:

```
00010 ENTRY CUST_INSERT
00020 REPLACE "ACCT:CUST_INSERT/DEPT1234/CUSTOMER RECORD INSERT PROGRAM"
00030 CSIOPTNS"COMMENTS=NO:FORCE=YES:SEQUENCE 5,5"
.
.
.
00520 EXIT
00530 COMPONENT"ACCT:CUS_INIT_FILE_HEADER"
00540 COMPONENT"ACCT:CUS_ERROR_PROC"
00550 COMPONENT"ACCT:CUS_TERMINATE"
```

---

## CURSOR

The CURSOR function indicates whether the cursor appeared in a specific field at the last terminal I/O. MANTIS makes the test and returns either TRUE or FALSE, or returns the screen or field name of the entity that contained the cursor at the last converse.

---

```
CURSOR( { screen - name, { field - name }
          { "FIELD"      { (row,col) }
          { "SCREEN"     } } )
```

---

### *screen-name*

<b>Description</b>	<i>Required.</i> Specifies the name (as defined in a previously executed SCREEN statement) of the screen you are testing.
<b>Format</b>	A MANTIS symbolic name (see “ <a href="#">Symbolic names</a> ” on page 24)

### *field-name*

<b>Description</b>	Specifies the name of the field you are testing.
<b>Format</b>	A MANTIS symbolic name (see “ <a href="#">Symbolic names</a> ” on page 24)

### *(row,col)*

<b>Restriction</b>	Your Master User determines if you have access to this option.
<b>Description</b>	<i>Optional.</i> Specifies the coordinates of the field within the logical display whose attributes you want returned.
<b>Format</b>	Numeric expressions that evaluate between 1 and 255
<b>Consideration</b>	The row and column positions must fall within a field in the specified map and within valid fields or MANTIS issues an error message.

## "FIELD"

**Description** Specifies that MANTIS should return the symbolic name of the field in which the cursor appeared at the last terminal input.

**Format** Text expression evaluating to "FIELD"

### Considerations

- ◆ This form of the CURSOR function returns a text string containing the field symbolic name.
- ◆ If the cursor was not in any named (input) field at terminal input, a zero-length text string is returned.

---

## "SCREEN"

**Description** Specifies that MANTIS should return the symbolic name of the screen containing the field in which the cursor appeared at the last terminal input.

**Format** Text expression evaluating to "SCREEN"

### Considerations

- ◆ This form of the CURSOR function returns a text string containing the screen symbolic name.
- ◆ If the cursor was not in any named (input) field at terminal input, a zero-length text string is returned.

### General considerations

- ◆ MANTIS returns TRUE only if the cursor appeared within the domain of the field you specified when ENTER, a program function key, a PA key, the PEN key, or the CLEAR key was pressed. Otherwise, MANTIS returns FALSE.
- ◆ If *screen-name* is defined in a caller and passed as a parameter on an EXTERNAL DO, both *screen-name* and *field-name* must be passed.
- ◆ MANTIS returns FALSE when the map is not in the current map set.
- ◆ MANTIS returns FALSE when a field was not displayed due to an overlaying map on the last CONVERSE operation.
- ◆ MANTIS returns TRUE if the cursor was in a field at least partially displayed in a map set.
- ◆ After you issue a CLEAR statement, the CURSOR function returns FALSE until the next CONVERSE.
- ◆ See also “**ATTRIBUTE**” on page 95 and “**MODIFIED**” on page 332.

### Examples

- ◆ For the table below, the cursor was on FIELD1 (at row 3, column 2), in screen MAP1, when the user pressed ENTER.

Example	Results	Comments
CURSOR(MAP1, FIELD1)	TRUE (1)	
CURSOR(MAP1, FIELD2)	FALSE (0)	
CURSOR(MAP1, (3, 2))	TRUE (1)	
CURSOR("SCREEN")	"MAP1"	Text version of the symbolic name.
CURSOR("FIELD")	"FIELD1"	

- ◆ The following example shows how to use CURSOR to determine if the location of the cursor during the last terminal input was on a specific field:

```

00010 IF CURSOR(MAP, CUST_NO)
00020 .DO CUST_NO_DRILLDOWN
00030 END

```

- ◆ The following example shows how to use CURSOR to select items from a list of options by cursor positioning. This requires specifying a symbolic map name and field name. For example, select an item from a list of choices (menu processing) or process an element from a list:

```
00100 WHEN CURSOR(MENU_SCREEN,CLIENT_UPDATE)
00110 .DO CLIENT_PROCESSING
00115 .BREAK
00120 WHEN CURSOR(MENU_SCREEN,INVENTORY_UPDATE)
00130 .DO INVENTORY_PROCESSING
00140 .BREAK
.
.
.
00200 END
```

- ◆ The following example shows how to use CURSOR to position the cursor to a group of lines. For example, MANTIS scrolls up or down when the cursor is the scroll amount in the Full Screen Editor:

```

00050 I =1
00100 WHILE I<=COUNT
00110 .IF CURSOR(CLIENT_SCREEN,NAME(I))
00120 ..FIRST_KEY=NAME(I)
00130 ..I=COUNT:|END LOOP FORCED
00140 .END
00150 .I=I+1
00160 END
00170 GET REC(FIRST_KEY)LEVEL=1
00180 I=1
00190 WHILE I<COUNT AND REC<>"END"
00200 .I=I+1
00210 .GET REC LEVEL=1
00220 END

```

- ◆ The following example shows how the “FIELD” form of the CURSOR function can be used to issue field sensitive help:

```

00100 SCREEN MAP(CLIENT_SCREEN)
00110 CONVERSE MAP
00120 WHEN MAP="PF1"
00130 .PROMPT CURSOR("FIELD")
00140 END

```

# DATAFREE

The DATAFREE function returns the number of bytes remaining in your data area. The data area is used to hold the values for all variables in your program. The amount available decreases as additional variables are defined.

## DATAFREE

### General consideration

See also “PROGFREE” on page 372 and “USERWORDS” on page 498.

**Example** The following example shows how to use the DATAFREE function:

Example	Results	Comments
DATAFREE	65535	For a program that has not been RUN or BOUND yet (no variables defined).



---

## DATE (Function)

DATE is both a statement and a function. The DATE function returns a text string containing the current date in the format of the current specification.

---

### DATE

---

#### General considerations

- ◆ The format of the date string can be an installation-defined value or a program-defined value (see “[DATE \(Statement\)](#)” on page 179). The supplied default is “YY/MM/DD”. The DATE statement and function provide flexibility for choice of the delimiter that appears between the elements of the DATE text string. You may choose no delimiter or any delimiter, such as a slash, a hyphen, or a period. MANTIS will return the text string in whatever format you choose.
- ◆ The DATE function may have substring parameters (see “[Substringing text variables](#)” variables on page 53).
- ◆ See “[DATE \(Statement\)](#)” on page 179 for information on specifying the format.
- ◆ See also “[TIME \(Function\)](#)” on page 460.

**Example**      The following examples show how the DATE function returns the current date in the format set up by the DATE statement:

Example	Results	Comments
DATE	"01/12/31"	Default format.
DATE	"12-31-2001"	Alternate format set up by: DATE="MM-DD-YYYY"
DATE(1,5)	"01/12"	Substringing allowed; in this case, just the YY/MM.
DATE	"01-02"	When DATE="YY-MM".
DATE	"2009/12/31"	When DATE="2009/12/31". All digits are considered punctuation characters. Subsequent DATE functions return this value until the format reset.

---

## DATE (Statement)

DATE is both a statement and a function. Use the DATE statement to specify a text string by which the DATE function formats the current date.

---

**DATE=mask-expression**

---



---

### *mask-expression*

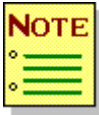
**Description**     *Required.* Specifies the type of string MANTIS uses to return the current date with the DATE function.

**Format**            A text string of 0–12 characters

### **Considerations**

- ◆ If the text string specified is longer than 12 characters, the first 12 bytes are used.
- ◆ If NULL ("" ) is specified, the installation default is used.
- ◆ The following strings are substituted with the data described when used with the DATE function:
  - *DD* or *dd*            Day of month (01–31)
  - *MM* or *mm*            Month (01–12)
  - *YY* or *yy*            2-digit year (00–99)
  - *DDD* or *ddd*          Julian day (001–366)
  - *YYYY* or *yyyy*       4-digit year (0000–9999)
  - Punctuation characters

## General considerations



---

Your Master User may define an installation-wide default value for the DATE format.

---

- ◆ You can specify the format of the DATE returned value in one of the following ways:
  - DATE=*mask*, as specified in a program by a user.
  - DATE format specified by installation (see your Master User).
  - Supplied default, which is “YY/MM/DD”.
- ◆ The format is maintained down DO/CHAIN levels.
- ◆ No left-hand subscripting is permitted. (DATE(1,5)=“YY/MM” (the DATE statement) is invalid. However, you can use subscripts on the DATE function.
- ◆ When MANTIS executes a CHAIN (without LEVEL), KILL, fault (error), or STOP, the format is reset to the installation default.
- ◆ YYYYMMDD is used for all CONTROL users programs.
- ◆ Arguments for DATE are converted to uppercase upon execution of your program.
- ◆ During testing, a constant DATE can be set; for example, DATE=“2009/12/31”. It is maintained according to the rules stated above.
- ◆ The DATE statement and function provide flexibility for choice of the delimiter that appears between the elements of the DATE text string. You may choose no delimiter, a slash, a hyphen, or a period. MANTIS will return the text string in whatever format you choose.
- ◆ See also “[TIME \(Statement\)](#)” on page 462 and “[DATE \(Function\)](#)” on page 177.

### Example

See the examples under “[DATE \(Function\)](#)” on page 177.

---

## DBCS (Statement)(Kanji users only)

The DBCS statement names and specifies dimensions for DBCS (Double Byte Character Set) variables and arrays.

---

**DBCS *name1*[(*n1*,] *length1*)]**

***[,name2*[(*n2*,] *length2*)] . . .]**

---

### *name*

**Description**     *Required.* Specifies the name of the DBCS variable.

**Consideration** When the symbolic name is previously defined, MANTIS bypasses this definition.

---

### *n*

**Description**     *Optional.* Specifies the number of elements in a DBCS array.

**Format**            Arithmetic expression that evaluates to a value in the range 1–255

#### **Considerations**

- ◆ MANTIS rounds *n* to an integer value.
- ◆ If not specified, *name1* is a DBCS scalar.

---

### *length*

**Description**     *Optional.* Specifies the maximum length (in characters) of each DBCS element.

**Format**            Arithmetic expression that evaluates to a value in the range 1–127

**Default**            8

**Consideration** MANTIS rounds *length* to an integer value.

## General considerations

- ◆ A DBCS variable contains a zero-length string (K"") upon initial definition.
- ◆ MANTIS accepts only as many characters in a DBCS variable as you specify in the DBCS statement.
- ◆ The following MANTIS statements allow DBCS literals and variables to be specified as arguments:

CALL	HEAD	POINT
DEQUEUE		SHOW
ENQUEUE	LET	SIZE
GET		

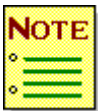
- ◆ The following MANTIS statements allow DBCS variables, but not DBCS literals, to be specified as parameters:

CHAIN	ENTRY-EXIT	DO	OBTAIN
-------	------------	----	--------

- ◆ The DBCS statement is functionally equivalent to the KANJI statement.
- ◆ See also "BIG" on page 134, "KANJI (Kanji users only)" on page 298, "MIXD" on page 327, "MIXM" on page 328, "MIXMODE" on page 329, "MIXT" on page 331, "SMALL" on page 415, and "TEXT" on page 457.

## Example

The following example shows how a DBCS statement names and specifies dimensions for DBCS variables:



In this example, < indicates SO or Shift-out, and > indicates SI or Shift-in.

```
00010 DBCS FIELDK(5),ARRAYK(3,20)
00020 FIELDK=K" %% ":ARRAYK(1)=K" %%% " :ARRAYK(2)=G "<%%%">"
00030 SCREEN MAP("DBCS_MAP", "PSW")
00040 WHILE MAP<>"CANCEL"
00050
00060
00070
.
```

---

## DELETE

The DELETE statement deletes a record from an external file, a MANTIS file, a personal computer file, an RDM logical view, or a TOTAL DBMS view. Before you delete a record from a file or view, you must first identify it by processing the associated FILE, TOTAL, ACCESS, or VIEW statement. You do not need to GET a MANTIS, TOTAL, or external file record before deleting it. You must, however, read an RDM logical view before deleting it.

### DELETE (External file)

---

```
DELETE file - name [ ( key1, key2 . . . ) ALL  
                     LEVEL = n  
                     ALL ]
```

---

---

#### *file-name*

**Description**     *Required.* Specifies the name (as defined in a previously executed ACCESS statement) of an existing external file where you want to delete a record.

**Format**            A MANTIS symbolic name (see “[Symbolic names](#)” on page 24)

**key1,key2,...**

**Description**     *Optional.* Specifies **generic** key values for *file-name*.

**Considerations**

- ◆ MANTIS deletes all records matching the specified key value(s).
- ◆ If you do not specify (*key1,key2,...*), MANTIS uses the current contents of the file key variables to locate and delete a single record.
- ◆ The ALL keyword is required if you specify (*key1,key2,...*).
- ◆ The datatypes of the supplied generic keys (*key1,key2,...*) must match the datatypes (text or numeric) of the corresponding elements in the file definition.
- ◆ If you supply complete key values, MANTIS deletes that specific record and returns a status with an FSI message of 1 DELETED RECORD. If the underlying file is a path with a non-unique alternate index, MANTIS will delete all synonym records that match the key specification.
- ◆ If the last (or only) key supplied is TEXT or DBCS, MANTIS finds any record whose key matches up to the current length of that TEXT or DBCS field and deletes the found record or records. In this case, the length of the generic key is the full length of all preceding key fields (if any) plus the current length of the last key field. For example, if the key field is TEXT and has a length of five, and *key* has the value of "ABC", MANTIS deletes any record with "ABC" in the first three key positions
- ◆ For TEXT or DBCS keys that are not in the last position, whose values are shorter than the corresponding file element length, MANTIS uses blanks to pad up to the length of the corresponding file element.
- ◆ If a TEXT or DBCS key is longer than the corresponding file element, MANTIS truncates the key to the length of the corresponding file element.
- ◆ MANTIS uses numeric keys in their entirety. For example, if the key value is 1, MANTIS will match only keys with 1 and not keys with 10–19, 100–199, etc.



---

**ALL**

**Description**     *Optional.* Tells MANTIS to delete all records in the file that match the key specification (*key1,key2,...*).

**Considerations**

- ◆ Specifying the following will delete all of the records in a file:

```
DELETE file-name ALL
```

- ◆ Specifying the following will delete all of a file's records that start with the specified generic key value:

```
DELETE file-name (key1, key2...) ALL
```

---

**LEVEL=*n***

**Description**     *Optional.* Specifies the buffer number that contains the record you want to delete.

**Default**            1

**Format**            Arithmetic expression that evaluates to a value in the range of 1 through *m*, where *m* is the maximum buffer number, as defined in the corresponding ACCESS statement

**Considerations**

- ◆ MANTIS uses only the integer portion of *n*.
- ◆ Only specify LEVEL=*n* when the *file-name* has buffers defined.
- ◆ Do not use the LEVEL option when issuing a generic DELETE.

General considerations

- ◆ The following examples show how these rules generate keys for a file with two five-character TEXT key fields:

Example	Key value used	Comments
DELETE FNAME	Current contents of the two key field variables	Specific key.
DELETE FNAME ALL	" "	Deletes all records in the file.
DELETE FNAME ( "12345" , "ABC" ) ALL	"12345ABC"	Generic key, length=8.
DELETE FNAME ( "12345" , "ABCDE" ) ALL	"12345ABCDE"	Full key (at most one record deleted).
DELETE FNAME ( "123" , "ABC" ) ALL	"123 ABC"	key1 blank padded. Generic key, length=8.
DELETE FNAME ( "1234567" , "ABC" ) ALL	"12345ABC"	key1 truncated. Generic key, length=8.
DELETE FNAME ( "12345" ) ALL	"12345"	Generic key, length=5.
DELETE FNAME ( "123" ) ALL	"123"	Generic key, length=3.

- ◆ MANTIS returns a text string in the variable called *access-name* that reflects the status of the operation:

Returned text string	Description
""	The delete is successful.
"LOCK" *	The password specified in the ACCESS statement for this file view is not valid for deletions or insertions.
"ERROR" *	MANTIS received an error status. Use the FSI function (see “FSI” on page 232) for more information. Possible causes: <ul style="list-style-type: none"> <li>◆ A physical error occurred during record deletion.</li> <li>◆ RRN for NUMBERED files specified a record number that does not exist.</li> <li>◆ The External file exit canceled the operation.</li> </ul>
"NOTOPEN" *	The external file is not open.

\* Returned only when TRAP is in effect for the file.

- ◆ If key(s) are not supplied on the delete statement for NUMBERED files, the Relative Record Number (RRN) contained in the corresponding reference variable identifies the record to be deleted.
- ◆ You cannot do a generic delete for NUMBERED files, but you can delete ALL, or delete a single record, the Relative Record Number (RRN) contained in the corresponding reference variable identifies the record to be deleted.
- ◆ If key(s) are not supplied on the DELETE statement for INDEXED files, the contents of key data elements identify the record to be deleted.
- ◆ If you do not specify keys, MANTIS deletes the record that has keys equal to the current contents of the key variable(s) for this file. It uses the level, if supplied, to determine the key subscript.

- ◆ If TRAP is not in effect, and you are unable to perform the delete because of a failure status from a MANTIS external file, MANTIS automatically issues a RESET. If trap is in effect, and the program does not issue a RESET when “ERROR” is returned, then it is possible that MANTIS did only part of the deletion.
- ◆ An external file open is issued (when required) on the first DELETE, GET, INSERT, or UPDATE.
- ◆ If a generic DELETE fails after partial completion, all updates are backed out if a fault message is issued, but, if TRAP is on, the actual number of deleted records is returned as part of the FSI message.
- ◆ DELETE with ALL option deletes all records in an external KSDS indexed file that matches the key qualification. The generic DELETE is not available on RRDS files, but DELETE ALL is available.
- ◆ Delete is not allowed for certain file types, for example, VSAM ESDS.
- ◆ For extended external file status messages and Function Status Indicators (FSIs), see “[Extended status messages for MANTIS and external files](#)” on page 521.
- ◆ The External File Exit can affect this statement. See your Master User for details.
- ◆ See also “[FSI](#)” on page 232, “[GET](#)” on page 234, “[INSERT](#)” on page 277, and “[UPDATE](#)” on page 479.

### Example

The following example shows an External File DELETE. Notice that a GET statement, which retrieves the designated record, precedes the DELETE statement:

```
00020 .ACCESS RECORD( "INDEX", "SERENDIPITY", 16)
00030 .SCREEN MAP( "INDEX" )
00040 .CONVERSE MAP
00050 .COUNTER=1
00060 .WHILE MAP<>"CANCEL" AND COUNTER<17
00070 ..WHEN INDICATOR(COUNTER)="G"
00080 ...GET RECORD LEVEL=COUNTER
00090 ..WHEN INDICATOR(COUNTER)="D"
00100 ...DELETE RECORD LEVEL=COUNTER
.
```

---

## DELETE (MANTIS file)

---

```
DELETE file - name [ (key1, key2 . . .) ALL  
                    LEVEL = n  
                    ALL ]
```

---

---

### *file-name*

**Description**     *Required.* Specifies the name (as defined in a previously executed FILE statement) of a file where you want to delete a record.

**Format**            A MANTIS symbolic name (see “[Symbolic names](#)” on page 24)

---

### *key1, key2, ...*

**Description**     *Optional.* Specifies the generic key value(s) and deletes all records beginning with the specified key value(s).

### **Considerations**

- ◆ If the last (or only) key supplied is TEXT or DBCS, MANTIS deletes any record matching the key up to the current length of the TEXT or DBCS field. For example, if the key is TEXT, has a length of 5, and has a value of “ABC”, 3 is used in the key length calculation.
- ◆ Datatypes of the supplied generic keys (*key1, key2, ...*) must match the text or numeric datatypes of the corresponding elements in the file definition.
- ◆ Specify ALL with this option to delete all records that match a generic key.
- ◆ For supplied keys that are TEXT, are not in the last position, and are shorter than the corresponding file element length, MANTIS blank-pads the field to the defined length.
- ◆ If a TEXT key is longer than the corresponding file element length, the text key is truncated.
- ◆ Substringing is not allowed for numeric generic key values. Numeric keys are used in full.

## ALL

**Description**     *Optional.* Tells MANTIS to delete all records in the file, or all records matching the specified key(s).

### Considerations

- ◆ Specifying the following will delete all of the records in a file:
- ◆ Specifying the following will delete all of a file's records that start with the specified generic key value:

```
DELETE file-name ALL
```

```
DELETE file-name (key1, key2...) ALL
```

---

## LEVEL=*n*

**Description**     *Optional.* Specifies the buffer number that contains the record you want to delete.

**Default**            1

**Format**            Arithmetic expression that evaluates to a value in the range of 1 through *m*, where *m* is the maximum buffer number, as defined in the corresponding FILE statement

### Considerations

- ◆ MANTIS uses only the integer portion of *n*.
- ◆ Only specify LEVEL=*n* when the *file-name* has buffers defined.
- ◆ Do not use the LEVEL option when issuing a generic DELETE.

## General considerations

- ◆ The following examples show how these rules generate keys for a file that has two five-character TEXT key fields:

Example	Key value used	Comments
DELETE FNAME	Current contents of the two key field variables.	Specific key.
DELETE FNAME ALL	" "	Deletes all records in the file.
DELETE FNAME ( "12345" , "ABC" ) ALL	"12345ABC"	Generic key, length=8.
DELETE FNAME ( "12345" , "ABCDE" ) ALL	"12345ABCDE"	Full key (at most one record deleted).
DELETE FNAME ( "123" , "ABC" ) ALL	"123 ABC"	key1 blank-padded. Generic key, length=8.
DELETE FNAME ( "1234567" , "ABC" ) ALL	"12345ABC"	key1 truncated. Generic key, length=8.
DELETE FNAME ( "12345" ) ALL	"12345"	Generic key, length=5.
DELETE FNAME ( "123" ) ALL	"123"	Generic key, length=3.

- ◆ MANTIS returns a text string in the variable called *file-name* that reflects the status of the operation:

Returned text string	Description
""	The delete is successful.
"LOCK" *	The password specified in the FILE statement is not valid.
"ERROR" *	MANTIS received an error status. Use the FSI function (see "FSI" on page 232) for more information. Possible causes: <ul style="list-style-type: none"> <li>◆ A physical error occurred during record deletion.</li> <li>◆ The SETPRAY Exit canceled the operation.</li> </ul>
"NOTFOUND"	A record with the requested key does not exist.

\* Returned only when TRAP is in effect for the file.

- ◆ If TRAP is not in effect and you are unable to perform the delete because of a failure status, MANTIS automatically issues a RESET. If TRAP is in effect and the program does not issue a RESET when "ERROR" is returned, then it is possible that MANTIS did only part of the deletion.
- ◆ If you do not specify keys, MANTIS deletes the record that has keys equal to the current contents of the *key* variable(s) for this file. It uses the level, if supplied, to determine the key subscript.
- ◆ If a numeric (BIG or SMALL) key is supplied, it must match exactly in the corresponding positions of the record.
- ◆ The number of records deleted for DELETE with the ALL option is returned in the FSI.
- ◆ If a full key value is supplied as a generic key value, MANTIS returns a GOOD status with an FSI message of 1 DELETED RECORD(S).
- ◆ The Setpray Exit can affect this statement. See your Master User for details.



- ◆ For extended MANTIS file status messages and Function Status Indicators (FSI), see “[Extended status messages for MANTIS and external files](#)” on page 521.
- ◆ See also “[FSI](#)” on page 232, “[GET](#)” on page 234, “[INSERT](#)” on page 277, and “[UPDATE](#)” on page 479.

### Example

The following example shows how a MANTIS File DELETE can be done in conjunction with a function key:

```

00010 ENTRY INDEX
00020 .FILE RECORD( "INDEX" , "SERENDIPITY" )
00030 .SCREEN MAP( "INDEX" )
00040 .GET RECORD
00050 .WHILE RECORD<>"END" AND MAP<>"CANCEL"
00060 ..CONVERSE MAP
00070 ..WHEN MAP="PF1"
00080 ...INSERT RECORD
00090 ..WHEN MAP="PF2"
00100 ...DELETE RECORD
00110 ..WHEN MAP="PF3"
00120 ...UPDATE RECORD
00130 ..END
00140 ..GET RECORD
00150 .END
00170 EXIT

```

## DELETE (Personal computer file)

---

**DELETE** *file-name*[LEVEL=*n*]

---

---

***file-name***

- |                    |   |
|--------------------|---|
| <b>Description</b> | <i>Required.</i> Specifies the name (as defined in a previously executed ACCESS statement) of a personal computer file where you want to delete a record. |
| <b>Format</b>      | A MANTIS symbolic name (see “ <a href="#">Symbolic names</a> ” on page 24)  |
- 

**LEVEL=*n***

- |                    |  |
|--------------------|--|
| <b>Description</b> | <i>Optional.</i> Specifies the buffer number that contains the record you want to delete.  |
| <b>Default</b>     | 1  |
| <b>Format</b>      | Arithmetic expression that evaluates to a value in the range of 1 through <i>m</i> , where <i>m</i> is the maximum buffer number, as defined in the corresponding ACCESS statement |

**Considerations**

- ◆ Only specify LEVEL=*n* when the *file-name* has buffers defined.
- ◆ MANTIS uses only the integer portion of *n*.

## General considerations

- ◆ You cannot delete records from SEQUENTIAL files.
- ◆ MANTIS returns a text string in the variable called *file-name* that reflects the status of the operation:

Returned text string	Description
""	The delete is successful.
"LOCK" *	The password specified in the ACCESS statement for this file view is not valid for deletions or insertions.
"ERROR" *	<p>MANTIS received an error status. Use the FSI function (see “FSI” on page 232) for more information. Possible causes:</p> <ul style="list-style-type: none"> <li>◆ A physical error occurred during record deletion.</li> <li>◆ RRN for NUMBERED files specified a record number that does not exist.</li> </ul>

\* Returned only when TRAP is in effect for the file.

- ◆ For NUMBERED files, the Relative Record Number (RRN) contained in the corresponding reference variable identifies the record to be deleted.
- ◆ File views residing on personal computer files can be accessed only by that computer user.
- ◆ For extended personal computer file status messages and Function Status Indicators (FSIs), see “[Extended status messages for MANTIS and external files](#)” on page 521.
- ◆ See also “FSI” on page 232, “GET” on page 234, “INSERT” on page 277, and “UPDATE” on page 479.

### Example

The following example shows how a Personal Computer File DELETE is coded with an ACCESS statement preceding the corresponding DELETE:

```
00020 .ACCESS RECORD("INDEX","SERENDIPITY",16)
00030 .SCREEN MAP("INDEX")
00040 .CONVERSE MAP
00050 .COUNTER=1
00060 .WHILE MAP<>"CANCEL" AND COUNTER<17
00070 ..WHEN INDICATOR(COUNTER)="G"
00080 ...GET RECORD LEVEL=COUNTER
00090 ..WHEN INDICATOR(COUNTER)="D"
00100 ...DELETE RECORD LEVEL=COUNTER
.
.
.
```

---

## DELETE (RDM logical view)

---

**DELETE** *view-name* [ALL][LEVEL=*n*]

---

---

### *view-name*

<b>Description</b>	<i>Required.</i> Specifies the name (as defined in a previously executed VIEW statement) of the logical view where you want to delete a logical record.
<b>Format</b>	A MANTIS symbolic name (see “ <a href="#">Symbolic names</a> ” on page 24)

---

### ALL

<b>Description</b>	<i>Optional.</i> Deletes all logical view records that are retrieved by automatically generated GET...NEXT statements.
<b>Format</b>	Must be coded exactly as shown

---

### LEVEL=*n*

<b>Description</b>	<i>Optional.</i> Specifies the buffer number that contains the record you want to delete.
<b>Default</b>	1
<b>Format</b>	Arithmetic expression that evaluates to a value in the range of 1 through <i>m</i> , where <i>m</i> is the maximum buffer number, as defined in the corresponding VIEW statement

### Considerations

- ◆ Only specify LEVEL=*n* when the *view-name* has buffers defined.
- ◆ MANTIS uses only the integer portion of *n*.

General considerations

- ◆ For RDM logical views, you must execute a corresponding VIEW statement and retrieve the logical record by using the GET...ENQUEUE statement before you can use the DELETE statement.
- ◆ Because logical views cannot be uniquely keyed, establish your current record position in a logical view (by reading the record) before you execute a DELETE.
- ◆ The RDM logical view DELETE view-name ALL statement deletes all records that were retrieved by a prior GET...NEXT statement.
- ◆ MANTIS returns a text string in the variable called *view-name* that reflects the status of the operation:

Returned text string	Description
""	Delete was successful.
"LOCK" *	You do not have permission to delete logical records from the logical view.
"NOTFOUND" *	The variable-entry chain set with the requested key does not exist.
"ERROR" *	MANTIS received an error status. Use the FSI function (see "FSI" on page 232) for more information. Possible causes: <ul style="list-style-type: none"><li>◆ An RDM error occurred during database access.</li><li>◆ You tried to perform an invalid function on the user view.</li></ul>

\* Returned only when TRAP is in effect for the file.

- ◆ If TRAP is not in effect and you are unable to perform the delete because of a failure status from RDM logical view, MANTIS automatically issues a RESET. If TRAP is in effect and the program does not issue a RESET when “ERROR” is returned, then it is possible that MANTIS does only part of the deletion.
- ◆ RDM logical view DELETE sends three status functions to the application program that indicate processing results—FSI, ASI, and VSI. FSI indicates the success or failure of your command. ASI indicates the status of each field in the logical record. VSI indicates the highest field status within the logical record. For a complete discussion of these status functions, see this chapter and “[Status functions](#)” on page 517.
- ◆ Your DBA can disallow deletes. If so, MANTIS returns the “LOCK” status if TRAP is in effect. If TRAP is not in effect, MANTIS displays a message and halts execution.
- ◆ MANTIS automatically issues a COMMIT prior to any terminal I/O if you have issued a DELETE since the last terminal I/O.
- ◆ For extended RDM view status messages and Function Status Indicators (FSIs), see “[Status functions](#)” on page 517.
- ◆ See also “[ASI](#)” on page 93, “[FSI](#)” on page 232, “[GET](#)” on page 234, “[INSERT](#)” on page 277, “[UPDATE](#)” on page 479, and “[VSI](#)” on page 505.

### Example

The following example shows how an RDM Logical View DELETE is coded. Notice that a corresponding VIEW statement is executed, and the view is read before the DELETE statement.

```

00010 VIEW CUSTOMER( "CUST" )
00020 SHOW "ENTER CUSTOMER NUMBER: "
00030 OBTAIN CUST_NO
00040 GET CUSTOMER(CUST_NO)
00050 IF CUSTOMER="FOUND"
00060 .DELETE CUSTOMER
00070 .SHOW "CUSTOMER DELETED"
00080 ELSE
00090 .SHOW "CUSTOMER NOT FOUND"
00100 END

```

## DELETE (TOTAL file view)

DELETE *file-name*[LEVEL=*n*]

### *file-name*

<b>Description</b>	<i>Required.</i> Specifies the name (as defined in a previously executed TOTAL statement) of an existing TOTAL file view where you want to delete a record.
<b>Format</b>	A MANTIS symbolic name (see “ <a href="#">Symbolic names</a> ” on page 24)

### LEVEL=*n*

<b>Description</b>	<i>Optional.</i> Specifies the buffer number that contains the record you want to delete.
<b>Default</b>	1
<b>Format</b>	Arithmetic expression that evaluates to a value in the range of 1 through <i>m</i> , where <i>m</i> is the maximum buffer number, as defined in the corresponding TOTAL statement

### Considerations

- ◆ Only specify LEVEL=*n* when the *file-name* has buffers defined.
- ◆ MANTIS uses only the integer portion of *n*.



## General considerations

- ◆ You must execute a corresponding TOTAL statement before the DELETE statement.
- ◆ A TOTAL file open is issued on the first DELETE, GET, INSERT, or UPDATE.
- ◆ You do not need to read a TOTAL file record before deleting it. MANTIS deletes the record that has keys equal to the current contents of the *key* variable(s) for this file. MANTIS uses the LEVEL, if supplied, to determine the key subscript.
- ◆ MANTIS obtains exclusive control of a record before requesting that TOTAL delete it. Therefore, you don't need to obtain the record using the ENQUEUE parameter on the GET statement (see “GET” on page 234).
- ◆ MANTIS returns a text string in the variable called *file-name* reflecting the status of the operation:

Returned text string	Description
""	The delete was successful.
"SETS" *	You tried to delete a master record while there were associated variable-entry chains.
"NOTFOUND" *	The variable-entry chain set with the requested key does not exist.
"LOCK" *	The password specified in the TOTAL statement is not valid for deletion.
"NOTOPEN" *	The TOTAL view is not open.
"NOTAVAL" *	The TOTAL file or view is not open.

\* Returned only when TRAP is in effect for the file.

- ◆ If TRAP is not in effect and you are unable to perform the delete because of a failure status from RDM logical view, MANTIS automatically issues a RESET. If TRAP is in effect and the program does not issue a RESET when “ERROR” is returned, then it is possible that MANTIS does only part of the deletion.
- ◆ See also “GET” on page 234, “INSERT” on page 277, and “UPDATE” on page 479.

**Example**

The following example shows how a TOTAL DELETE works. Notice a TOTAL statement is executed before the record is deleted.

```
00010 SCREEN MAP( "DELETE_HISTORY" )
00020 TOTAL CUSTOMERS( "CLIENT", "SALES" )
00030 TOTAL HISTORY( "PAYMENTS", "TEXAS", 11 )
00040 TEXT CUSTOMER_ID( 20 )
00050 SMALL BUFFER
00060 CUSTOMER_ID="OUR-BEST"
00070 GET CUSTOMERS( CUSTOMER_ID )
00080 BUFFER=1
00090 GET HISTORY SET( CUSTOMER_ID ) FIRST LEVEL=BUFFER
00100 WHILE HISTORY<>"END" AND BUFFER<11
00110 .IF PAY_DATE<"970821"
00120 ..DELETE HISTORY LEVEL=BUFFER
00130 .ELSE
00140 ..BUFFER=BUFFER+1
00150 .END
00160 .GET HISTORY SET( CUSTOMER_ID ) LEVEL=BUFFER
00170 END
00180 CONVERSE MAP
```

---

## DEQUEUE

The DEQUEUE statement releases control of a resource, or, for VSAM External file users, releases a previously reserved External file record, or for TOTAL users, releases a previously reserved TOTAL database record. MANTIS also releases any program or TOTAL record that is waiting while other tasks use a resource.

---

**DEQUEUE** { *resource* }  
                  { *file - name* }

---

---

### *resource*

<b>Description</b>	<i>Required.</i> Specifies the resource (text or DBCS expression) held by a previously executed ENQUEUE statement.
<b>Format</b>	Text or DBCS expression for a resource

---

### *file-name*

<b>Description</b>	<i>Required.</i> Specifies the name (as defined in a previously executed ACCESS or TOTAL statement) of a TOTAL view or External file view you want to release.
<b>Format</b>	A MANTIS symbolic name (see “ <a href="#">Symbolic names</a> ” on page 24) for a TOTAL or External file view

## General considerations

- ◆ Before executing the DEQUEUE statement, you must issue either an ENQUEUE statement against the same resource or a GET statement with ENQUEUE parameters for a TOTAL or external file view.
- ◆ If the resource is a TOTAL or external file view, MANTIS releases all records enqueued by the GET statement from the exclusive control of the current task.
- ◆ This statement is required only in very special circumstances for TOTAL view processing. Consult your Master User before you use DEQUEUE for TOTAL view processing.
- ◆ Use this statement when an external VSAM GET statement with ENQUEUE parameter has previously been issued and no update is done to the record.
- ◆ With DEQUEUE, there is a limited scope of applicability when running in different environments such as CICS and BATCH. When you are running in both, DEQUEUE will apply in one mode and not in the other.
- ◆ Batch MANTIS does not support DEQUEUE (it is ignored).
- ◆ MANTIS for IMS does not support the ENQUEUE, DEQUEUE, and COMMIT statements. To maintain compatibility with other MANTIS versions, these statements may remain in existing programs. MANTIS for IMS will ignore the statements.
- ◆ See also “ENQUEUE” on page 211 and the ENQUEUE parameter under the “GET” statements on page 234.

## Examples

- ◆ The following example shows how MANTIS uses a DEQUEUE statement to release the resources of a file called "CUSTOMERS". Notice that the record is ENQUEUED before the DEQUEUE statement.

```
00020 FILE REC("CUSTOMERS", "ALIBABA")
.
.
.
00100 ENQUEUE "CUSTOMERS"+RECORD_KEY
00110 GET REC(RECORD_KEY)
.
.
.
00200 UPDATE REC
00210 DEQUEUE "CUSTOMERS"+RECORD_KEY
```

- ◆ The following example shows how MANTIS uses a DEQUEUE statement to release the file record from a file called "CUSTOMERS". Notice that the record is ENQUEUED with a GET before the DEQUEUE statement.

```
00020 ACCESS REC ("CUSTOMERS", "ALIBABA")
.
.
.
00100 GET REC (RECORD_KEY) ENQUEUE (Logic determines no
update is necessary)
.
.
.
00200 DEQUEUE REC
```

## DO

The DO statement transfers program execution to an internal or external subroutine. An internal subroutine is a block of statements within the existing MANTIS program, marked by an ENTRY-EXIT. A PROGRAM statement identifies an external subroutine. An internal subroutine performs a function required at one or more points within a program. An external subroutine performs a function required by one or more programs. After executing the subroutine and upon encountering an EXIT statement, execution returns to the next program line following the DO statement. (“External DO” on page 527 contains a detailed discussion of the External Do function.)

---

**DO *entry-name*[(*argument1*,*argument2*,...)]**

---

---

### ***entry-name***

**Description**     *Required.* Specifies the name of a subroutine as indicated in the ENTRY or PROGRAM statement.

**Format**            A MANTIS symbolic name (see “Symbolic names” on page 24)

### **Considerations**

- ◆ *entry-name* must match a name defined in an ENTRY or PROGRAM statement.
- ◆ The argument(s) must be a previously defined, unsubscripted variable name (constants, literals and expressions are not allowed). This means that you can only pass an entire array with external DO, but *not* specific elements in an array (because you must subscript a variable to identify its position in an array).
- ◆ The argument(s) in the DO and ENTRY-EXIT statements must correspond in type and number.
- ◆ Because all variables previously defined by the calling routine are available to an *internal* subroutine, MANTIS uses arguments only to set up alias names.
- ◆ Any argument passed and then modified in a subroutine retains the modified value when execution returns following the DO.

---

**argumentn**

**Description**     *Optional.* Specifies the argument(s) you want passed to the subroutine.

**Format**            A MANTIS symbolic name (see “**Symbolic names**” on page 24)

**General considerations**

- ◆ You must include an ENTRY-EXIT statement around a subroutine.
- ◆ The first line of an externally done program must be an ENTRY statement.
- ◆ The DO statement must appear on a line by itself. MANTIS ignores any additional statements coded on the end of a DO statement (and separated with a colon).
- ◆ Only variables passed as arguments to an *external* subroutine (identified by a PROGRAM statement) are available to the external routine. If the variable is a SCREEN, FILE, VIEW, TOTAL, or ACCESS INTERFACE variable, subvariables are not available unless explicitly passed.
- ◆ In programming mode, if you stop during execution of an external subroutine, enter SHOW DOLEVEL to see where you are. Use the EXIT command to return to your calling routine.
- ◆ If you have a choice between conversing (or using any statement that issues a COMMIT) in high or low level External DOs, avoid low level DOs because the rollouts and rollins required may adversely affect performance.
- ◆ Avoid extensive modularizing in lower level External DOs because this can adversely affect performance.
- ◆ The SLICE and SLOT statements are ignored in an externally done program.
- ◆ The Program Load Exit can affect this statement. See your Master User for details.
- ◆ See also “**CHAIN**” on page 139, “**DOLEVEL**” on page 209, “**ENTRY-EXIT**” on page 213, “**EXIT**” on page 219, and “**PROGRAM**” on page 373.

**Example**

The following example shows how the DO statement accesses a subroutine:

```
00100 ENTRY EDIT_PROGRAM
      .
      .
      .
00150 .TYPE ="CREDIT CHECK"
00160 .PROGRAM EDIT_RTN( "VALIDATION", "COMMON" )
00170 .DO EDIT_RTN(TYPE,CUST_NO,STATUS,MESSAGE)
00180 .IF STATUS<>"GOOD"
00190 ..PROGRAM ERROR_RTN( "CHECK FIELDS", "COMMON" )
00200 ..DO ERROR_RTN(CUST_NO)
00210 .END
00220 .DO EDIT_RTN(TYPE,CUST_NO,STATUS,MESSAGE)
00230 .IF STATUS<>"GOOD"
00240 ..DO ERROR_RTN(SALES_REP)
00250 .ELSE
00260 ..SALES_REP=MESSAGE
00270 .END
00280 .STOP
00290 EXIT
00300 ENTRY ERROR_RTN(FIELD)
00310 .IF NOTE=" "
00320 ..NOTE=MESSAGE
00330 ..ATTRIBUTE(MAP,FIELD)="BRI,CUR"
00340 .ELSE
00350 ..ATTRIBUTE(MAP,FIELD)="BRI"
00360 .END
00370 EXIT
```



---

## DOLEVEL

The DOLEVEL function returns the current execution level in an external subroutine.

---

### DOLEVEL

---

#### General considerations

- ◆ Use the SHOW DOLEVEL command when you are debugging to see which level you are executing. You can modify and replace programs at any level. The revised version is executed in the next RUN.
- ◆ You can use DOLEVEL to direct execution of a program; for example, EXIT or CHAIN to another program. You can also use DOLEVEL during debugging to help identify your program level.
- ◆ See also “DO” on page 206, “ENTRY-EXIT” on page 213, “EXIT” on page 219, and “PROGRAM” on page 373.

#### Example

Example	Results	Comments
DOLEVEL	0	Top-level program
DOLEVEL	1	First called program

E

The E function returns the value of natural e (2.71828182845905).

E

General considerations

- ◆ Do not use E as a variable name (see “Symbolic names” on page 24).
- ◆ See also “EXP” on page 220 and “LOG” on page 320.

Example

Example	Results	Comments
E	2.71828183	Constant value

---

## ENQUEUE

The ENQUEUE statement holds control of a resource as identified by the resource-string. Any subsequent ENQUEUE on that resource by another program causes that program to remain in a wait state until the resource is released (see the DEQUEUE statement).

---

### ENQUEUE *text-expression*

---

#### *text-expression*

**Description**     *Required.* Specifies the resource-string you want to hold.

**Format**            Text or DBCS expression

#### General considerations

- ◆ An ENQUEUE takes a *text-expression* and ensures that no one else is ENQUEUEd on the same *text-expression*. If there is someone else already ENQUEUEd upon that *text-expression* the latter MANTIS program waits until the original ENQUEUE is released by DEQUEUE, COMMIT, or screen output. ENQUEUE only works if *all* programs wishing to serialize on a resource use the correct enqueueing protocol.
- ◆ Do not specify the symbolic file-name as the resource for the ENQUEUE. The ENQUEUE function will not work as intended if coded as:
 

```
100 ENQUEUE REC
```

 because REC will evaluate to the file status; for example, "NEXT".
- ◆ A terminal I/O with COMMIT ON releases all ENQUEUEs (see "VSAM deadlocks" on page 550 and COMMIT ON/OFF).
- ◆ It is possible to deadlock tasks, if Task A holds an ENQUEUE that Task B is waiting to ENQUEUE and Task A is waiting on an ENQUEUE that Task B holds. (It is also possible for more than two tasks to have an *n*-way deadlock.) Therefore, it is important that you establish an ordering convention (for example, alphabetical order) so that all ENQUEUEs are issued in the same order.
- ◆ Records that are updated can be held by the host DBMS or file management system until a logical unit of work (LUW) is completed.

- ◆ Excessive ENQUEUEing can exhaust the TP monitor's allocated space for ENQUEUE registration.
- ◆ With ENQUEUE, there is a limited scope of applicability when running in different environments such as CICS and BATCH. When you are running in both, ENQUEUE will apply in CICS and is ignored in BATCH.
- ◆ Batch MANTIS and MANTIS for IMS do not support the ENQUEUE, DEQUEUE, and COMMIT statements. To maintain compatibility with other MANTIS versions, these statements may remain in existing programs. MANTIS will ignore the statements.
- ◆ See also “COMMIT” on page 149, “DEQUEUE” on page 203, and the **ENQUEUE parameter** under the “GET” statement (the “GET” statement starts on page 234).

**Example**

The following example shows how the ENQUEUE statement is used for record management and data integrity:

```
00020 FILE REC("CUSTOMERS", "ALIBABA")
.
.
.
00100 ENQUEUE "CUSTOMERS"+RECORD_KEY
00110 GET REC(RECORD_KEY)
.
.
.
00200 UPDATE REC
00210 DEQUEUE "CUSTOMERS"+RECORD_KEY
```

## ENTRY-EXIT

The ENTRY-EXIT statement defines the boundary of a subroutine or top-level routine of a program. When a DO or CHAIN statement invokes a subroutine or program bounded by an ENTRY-EXIT, the arguments (and all references to them) passed by the DO or CHAIN statement replace the subroutine's parameters. EXIT is also a command in the editor.

---

**ENTRY** *entry-name* [(*parameter1,parameter2,...*)]

.  
.  
.  
**statements**

.  
**EXIT**

---

### *entry-name*

**Description**     *Required.* Specifies the name of the subroutine or program.

**Format**            A MANTIS symbolic name (see “[Symbolic names](#)” on page 24)

### **Considerations**

- ◆ Must appear on same program line as ENTRY statement.
- ◆ Must be a unique symbolic name within the program. The entry name is defined at the time the program is saved or replaced; that is, before any statements are executed. For example:

```
00010 BIG X
...
01000 ENTRY X
...
02000 EXIT
```

X will be defined as the entry name. Statement 10 will be ignored because X is an already-defined symbolic name.

**parameter**

**Description**     *Optional.* Specifies those parameters you want passed to the subroutine or program.

**Format**            A MANTIS symbolic name (see “[Symbolic names](#)” on page 24)

**Considerations**

- ◆ A limit of 255 parameters per ENTRY exists.
- ◆ Until the EXIT statement is executed, *parameters* are aliases for the corresponding symbolic names passed on the DO statement. If the symbolic name specified by *parameter* is already defined and has a data area, that definition is temporarily suspended. See “[Automatic mapping](#)” on page 541.
- ◆ The way that you specify parameters in complex statements such as ACCESS, FILE, INTERFACE, TOTAL, or VIEW could cause a loss of automapping. This loss occurs if a complex statement in an internal subroutine references a field that is defined by a different name outside of the parameter list in the internal DO. To avoid this loss of automapping do one of the following:
  - Define the parameter with the same variable name in and out of the internal subroutine.
  - Do not pass the variable to a subroutine that uses complex statements.
  - Execute the complex statement outside the subroutine.
  - An example of this would be:

```
SCREEN MAP("MAP1"): | DEFINES NAME, FIELD1, FIELD2
CONVERSE MAP
DO SUB1(FIELD1)
GET REC
ENTRY SUB1(NAME)
FILE REC(NAME,PASSWORD): | DEFINES FIELD3, NAME
EXIT
```

Within SUB1, NAME and FIELD1 will be synonymous. So the both the FIELD1 and NAME field from the file, REC, will map to FIELD1 on I/o operations outside of SUB1. The GET REC statement in the program mainline will not properly retrieve the data for the variable NAME.

---

## EXIT

**Description**     *Required.* Returns control from a subroutine to the invoking program, or from a program to programming mode (if initiated from programming mode).

### General considerations

- ◆ When you pass data from one program to another, put an ENTRY-EXIT statement around the top-level of routine of each program.
- ◆ Arithmetic, text, or DBCS variables, lists or arrays must correspond in number between the ENTRY-EXIT and the DO and CHAIN statements, except when zero arguments are passed as a CHAIN. Arguments are optional on the CHAIN, even when parameters are specified on the target program's ENTRY statement. If arguments are not supplied, the parameter variables are undefined at entry. If arguments are supplied on a CHAIN or DO, the parameter variables assume the type characteristics from the passed arguments.
- ◆ If a program has arguments passed to it, or if it is externally done, it must have an ENTRY statement in it. That ENTRY statement must be the first line of the program.
- ◆ If the program is not externally done or does not have arguments passed to it, it does not need an entry-exit around the top-level routine. In this case, you must code a STOP statement prior to the first internal subroutine ENTRY statement.
- ◆ See also “CHAIN” on page 139, “DO” on page 206, and “STOP” on page 452.
- ◆ The ENTRY and EXIT statements must each appear on a line by themselves. Only a comment (separated by a colon) can follow the EXIT. For example:

```
ENTRY ROOTS(A,B,C,R1,R2)
```

```
...
```

```
EXIT: |Return with value in R1 & R2
```

**Example**

The following example shows how ENTRY-EXIT statements work in pairs to frame subroutines:

```
00010 ENTRY DATA_ENTRY
00020 .SCREEN MAP( "INDEX" )
00030 .FILE REC( "INDEX" , "SERENDIPITY" )
00040 .CONVERSE MAP
00050 .WHILE MAP<>"CANCEL"
00060 ..DO INSERT_RECORD
00070 ..CLEAR MAP
00080 ..CONVERSE MAP
00090 .END
00100 .STOP
00110 EXIT
00120 ENTRY INSERT_RECORD
00130 .INSERT REC
00140 EXIT
```



---

## EXEC\_SQL-END

The EXEC\_SQL-END statement defines the boundary of an SQL statement embedded in a MANTIS program. This allows SQL statements to be executed in a MANTIS program.

---

### EXEC\_SQL [ (nn) ]

---

#### General considerations

- ◆ MANTIS SQL Support must be installed on your system to execute SQL statements from a MANTIS program. If MANTIS SQL Support is not available on your system, SQL statements can be included and displayed in a MANTIS program, but they cannot be executed.
- ◆ Only SQL statement text can be placed between the EXEC\_SQL and END. MANTIS program statements or comments cannot be included within the SQL statement text.
- ◆ Each line of SQL text must begin with the MANTIS comment character, |.
- ◆ SQL statement text can be included on the same line with the EXEC\_SQL statement. For example:

```
00010 EXEC_SQL:|SELECT T
00020 END
```

- ◆ When SUPRA SQL is being used as the SQL database, a numeric session number is permitted following the EXEC\_SQL statement. For example, this statement will access the second SUPRA session when the SQL statement is called:

```
00010 EXEC_SQL(2)
```

This session number is not permitted when DB2 is the SQL database.

- ◆ See also “[SQLCA \(Function\)](#)” on page 420, “[SQLCA \(Statement\)](#)” on page 424, “[SQLDA \(Function\)](#)” on page 426, “[SQLDA \(Statement\)](#)” on page 432.
- ◆ For further information, refer to *MANTIS DB2 Programming, OS/390, VSE/ESA*, P39-5028, or *MANTIS SUPRA SQL Programming, OS/390, VSE/ESA*, P39-3105.

**Example**

The following example shows how the EXEC\_SQL statement allows SQL functions to be inserted into a MANTIS program. The END statement signals the end of the SQL segment of code.

```
00110 BIG EMPL_NUM
00120 TEXT EMPL_NAME(30)
00130 EXEC_SQL
00140 .| SELECT EMPNO, EMPNAME
00150 .| INTO: :EMPL_NUM, :EMPL_NAME
00160 .| FROM TEMPL_TABLE
00170 .| WHERE EMPNO = :EMPL_NUM
00180 END
```

---

## EXIT (Command)

The EXIT command/statement returns control from an external routine to the invoking program.

---

### EXIT

---

#### General considerations

- ◆ In programming mode, EXIT (as a command) returns you to programming mode for the calling program.
- ◆ In a running program, the EXIT statement both bounds the subroutine started by the ENTRY statement and, when executed, returns to the line following the DO that invoked the subroutine.
- ◆ You cannot exit from the highest-level routine (that is, DOLEVEL 0) in programming mode.
- ◆ In a running program, the highest-level EXIT is equivalent to a STOP.
- ◆ See also “DO” on page 206, “DOLEVEL” on page 209, “ENTRY-EXIT” on page 213, and “STOP” on page 452.

#### Example

```
00120 ENTRY INSERT_RECORD  
00130 .INSERT REC  
00140 EXIT
```

EXP

The EXP function returns the value of natural (e) to the power of *a*, where *a* is any arithmetic expression.

EXP(*a*)

*a*

**Description**     *Required.* Specifies any valid arithmetic expression.

**General consideration**

See also “E” on page 210 and “LOG” on page 320.

**Example**             The following example shows how the EXP function works:

Example	Results	Comments
EXP(0)	1	
EXP(100)	.268811714E44	
EXP(-34)	.171390843E-14	
EXP(PI)	23.1406926	

# FALSE

The FALSE function is a MANTIS constant that returns the value zero.

## FALSE

### General considerations

- ◆ Any numeric expression that evaluates to zero can be considered FALSE for purposes of a logic statement (IF, UNTIL, WHEN, and WHILE). For example, the expression NUMBER\_OF\_ERRORS will evaluate to FALSE when NUMBER\_OF\_ERRORS is zero. Otherwise, it will evaluate to TRUE.
- ◆ See also “TRUE” on page 472 and “ZERO” on page 512.

### Example

The following example shows how the FALSE function can be used to test the validity of user input:

```

00010 ENTRY CUST_ENTRY
00020 .SCREEN MAP( "CUST_ENTRY" )
00030 .FILE REC( "CUST_FILE" ,PASSWORD)
00040 .FILE ST_CODES( "STATE_CODES" ,PASSWORD)
00050 .CONVERSE MAP
00060 .WHILE MAP<>"CANCEL"
00070 ..ERROR=FALSE
00080 ..DO VALIDATE_INFO
00090 ..IF NOT( ERROR)
00100 ...INSERT REC
00110 ...CLEAR MAP
00120 ..END

```

Example	Results	Comments
FALSE	0	
NOT( FALSE)	1	

---

# FILE

The FILE statement identifies a MANTIS internal file that your program accesses. MANTIS retrieves the file description from your library and places it in your work area.

---

```
FILE name1([library1:]file-name1,password1[,PREFIX][,n1])  
      [,name2([library2:]file-name2,password2[,PREFIX][,n2]) . . .]
```

---

---

## name

Description	<i>Required.</i> Specifies a name for the file that you use in subsequent GET, UPDATE, INSERT, and DELETE statements.
Format	A MANTIS symbolic name (see “Symbolic names” on page 24)
Consideration	When the symbolic name is previously defined, MANTIS bypasses this definition.

---

## [library:]file-name

Description	<i>Required.</i> Specifies the name of the file profile as it was saved during file design.
Format	1–33 character text expression that evaluates to a valid file profile name
Considerations	<ul style="list-style-type: none"><li>◆ MANTIS translates this expression to uppercase upon execution of your program.</li><li>◆ If the file is in another user’s library, you can access it by specifying the name of the user in whose library it does reside (<i>library:</i>). If this parameter is used, the colon (:) is required.</li><li>◆ If the file does reside in your library, you may supply the file view name only. If you want this entity to be HPO bound, the library name is required, even if it is your own library.</li></ul>

---

**password**

**Description**     *Required.* Specifies the password valid for the type of file access your program needs (e.g., read only, update, insert/delete).

**Format**            Text expression that evaluates to a valid password

**Considerations**

- ◆ MANTIS does **not** translate the *password* expression to uppercase upon execution of your program, and the *password* expression must match exactly one of the passwords assigned in file design.
- ◆ Insert/delete allows read and update actions.
- ◆ When you use the *n* parameter to indicate multiple buffers, you should also add the LEVEL=*n* option to GET, UPDATE, INSERT, and DELETE statements.

---

**PREFIX**

**Description**     *Optional.* Indicates that MANTIS places the symbolic name and an underscore before all field names associated with this file. For example, if you code:

```
FILE CUST( "CUSTOMER" ,PASSWORD,PREFIX)
```

...and the file CUSTOMER has fields NAME and ADDR\_LINE1, the MANTIS program would refer to those fields as CUST\_NAME and CUST\_ADDR\_LINE1.

**Format**            Must be coded exactly as shown

**Considerations**

- ◆ Inhibits auto-mapping of variables because field names with PREFIX do not usually match the field names set up in Screen Design or in other entities.

- ◆ You can use PREFIX to keep multiple views of the same Internal File and yet have independent variable values. For example, you may wish to:
  - Enter data on a screen and automap to a file.
  - Non-destructively read that same file to check for other current values.

In order to do both of these things, you can use a PREFIXed FILE view to read for current values but not affect the data entered on the screen. The following example shows how you can use the VALIDATE file to achieve this:

```
FILE REC( "CUSTOMER" ,PASSWORD)
FILE VALIDATE( "CUSTOMER" ,PASSWORD,PREFIX)
```

- ◆ You can use PREFIXed FILEs to have multiple, independent positions within a file. The following example shows how you can use the SHADOW file to achieve this:

```
FILE REC( "CUSTOMER" ,PASSWORD)
FILE SHADOW( "CUSTOMER" ,PASSWORD,PREFIX)
```

- ◆ You can use PREFIX in combination with *Level* to have both single-occurrence and multiple-occurrence views of a file. For example:

```
FILE DETAIL( "CUSTOMER" ,PASSWORD,PREFIX)
FILE BROWSE( "CUSTOMER" ,PASSWORD,PREFIX,20)
...
IF CURSOR(BROWSE_MAP,BROWSE_NAME(I))AND BROWSE_MAP="PF1"
.GET DETAIL (BROWSE_NAME(I))
.CONVERSE DETAIL_MAP
END
```

- ◆ You can use PREFIX when you do not want similarly-named fields to automap between the same or different entity types. For example:

```
FILE CUST( "CUSTOMER" ,PASSWORD,PREFIX)
FILE VENDOR( "VENDOR" ,PASSWORD,PREFIX)
FILE EMPL( "EMPLOYEE" ,PASSWORD,PREFIX)
...
IF CUST_NAME=VENDOR_NAME
...
IF EMPL_NAME=CUST_NAME
```



---

*n*

<b>Description</b>	<i>Optional.</i> Indicates how many buffers MANTIS should allocate to this file.
<b>Default</b>	1
<b>Format</b>	Arithmetic expression that evaluates to a value in the range 1 through <i>n</i> , where <i>n</i> is the maximum buffer number, as defined in the corresponding ACCESS statement

**Consideration** MANTIS uses only the integer portion of *n*.

### General considerations

- ◆ You cannot use LEVEL on GET, UPDATE, INSERT, and DELETE statements if LEVEL is omitted on the FILE statement.
- ◆ The Setpray Exit can affect this statement. See your Master User for details.
- ◆ See also “DELETE” on page 183, “FSI” on page 232, “GET” on page 234, “INSERT” on page 277, “UPDATE” on page 479, and “TRAP” on page 469.

**Example** The following example shows how MANTIS uses the FILE statement to access a file record:

```
00020 .FILE RECORD( "INDEX", "SERENDIPITY", 16 )
00030 .SCREEN MAP( "INDEX" )
00040 .WHILE RECORD<>"END" AND MAP<>"CANCEL"
00050 ..CLEAR MAP:BUFFER=1
00070 ..GET RECORD LEVEL=BUFFER
00080 ..WHILE RECORD<>"END" AND BUFFER<17
00090 ...BUFFER=BUFFER+1
00100 ...GET RECORD LEVEL=BUFFER
00110 ..END
00120 ..CONVERSE MAP
00130 .END
```

---

# FOR-END

Use the FOR-END statements to execute a block of statements repeatedly while a counter is incremented or decremented through a specified range of values.

---

```
FOR counter=initial TO final [BY increment]
    .
    statements
    .
END
```

---

---

## counter

Description	<i>Required.</i> Specifies the numeric variable or array element that MANTIS uses as a counter.
Format	A MANTIS symbolic name (see “ <a href="#">Symbolic names</a> ” on page 24)

---

## initial

Description	<i>Required.</i> Specifies the initial value of the counter.
Format	Arithmetic expression
Consideration	MANTIS evaluates the expression only before the first loop iteration.

---

## final

Description	<i>Required.</i> Specifies the final value with which the counter is compared.
Format	Arithmetic expression
Consideration	MANTIS evaluates the expression at each loop iteration.

**BY increment**

**Description** *Optional.* Specifies the increment to be added to the counter after each execution of the block of statements.

**Default** BY 1

**Format** Arithmetic expression

**Consideration** MANTIS evaluates the expression at each loop iteration.

**General considerations**

- ◆ MANTIS sets the counter to the initial value, which is evaluated only once. MANTIS evaluates the final value and the increment prior to each execution of the block of statements.
- ◆ MANTIS repeatedly executes the block of statements while the counter is less than or equal to the final value and the increment is positive, or the counter is greater than or equal to the final value and the increment is negative. After executing each block of statements, MANTIS adds the increment to the counter. The following table summarizes the loop condition tests after the first execution:

Add increment to counter, then if...	Counter>final	Counter=final	Counter<Final
Increment > 0	Terminate	Execute	Execute
Increment = 0	Terminate	Terminate	Terminate
Increment < 0	Execute	Execute	Terminate

- ◆ MANTIS will not execute the block of statements when the initial value exceeds the final value (when the increment is positive) or the initial value is less than the final value (when the increment is negative). The following summarizes the loop condition tests for first execution:

	Initial>final	Initial=final	Initial<Final
Increment > 0	Terminate	Execute	Execute
Increment = 0	Terminate	Terminate	Terminate
Increment < 0	Execute	Execute	Terminate

- ◆ If the block of statements changes the values of the counter, the final value, or the increment, this value is reflected in the next comparison of the counter and the final value.
- ◆ For greater efficiency, if the final value or increment are expressions that do not change within the loop, assign them to a symbolic name so that they do not need to be evaluated on each iteration of the loop. (See the following example using FINAL).
- ◆ See also “BREAK” on page 136, “IF-ELSE-END” on page 274, “NEXT” on page 335, “WHEN-END” on page 508, “WHILE-END” on page 510, and “UNTIL-END” on page 478.

**Example**

The following example shows the FOR-END statement using literals. Note that the statement within the loop is executed five times, and the loop counter (I) will be equal to 6 when statement 50 is executed.

```
10 FOR I = 1 TO 5 BY 1
20 .NUM(I)=I
40 END
50 A=NUM(3)
```

The following example shows the FOR-END statement using initial, final, and increment variables:

```
10 BIG I,INITIAL,FINAL,INCREMENT:TEXT STRING(100)
20 INITIAL=10
30 FINAL=SIZE(STRING,"MAX")
40 INCREMENT=10
50 FOR I=INITIAL TO FINAL BY INCREMENT
60 . (statement logic)
70 END
```

The following example shows the FOR-END using a negative increment:

```
10 FOR J=SIZE(STRING) TO 1 BY -1
20 .IF STRING(J,J)<>" " <-- If this statement evaluates to TRUE,
30 ..BREAK                                     then logic will continue at line 60.
40 .END
50 END
.
.
.
```

---

# FORMAT

The FORMAT function returns a text string conversion of a numeric expression according to the supplied edit mask. This function can be used to format numeric fields into text for database, SHOW, or any other text output. This function also allows you to test screen design masks for expected results.

---

**FORMAT(*a*, *mask* [*,digit-select-character*])**

---

***a***

<b>Description</b>	<i>Required.</i> Specifies the numeric expression you want to format with an edit mask.
--------------------	---

---

***mask***

<b>Description</b>	<i>Optional.</i> Specifies the edit mask.
<b>Format</b>	A 1-254 character text expression
<b>Consideration</b>	Blank characters in the mask must appear as blanks. The blank-fill character used in screen design (default is the vertical bar) is not required.

---

***digit-select-character***

<b>Description</b>	<i>Optional.</i> Specifies the character in the mask operand that is filled with digits from a numeric expression.
<b>Default</b>	Installation defined (distributed with #)
<b>Format</b>	A 1-character text expression containing the mask character

**General considerations**

- ◆ A discussion of valid edit masks appears in *MANTIS Facilities, OS/390, VSE/ESA*, P39-5001.
- ◆ Use the FORMAT function with the SHOW command to test your edit mask or to produce masked output on SHOWs.
- ◆ See also “*Designing screens*” (mask characters/formatting) on page 556 and “*TXT*” on page 473.

## Examples

Example	Results	Comments
<code>FORMAT(100,"\$\$#Z.##")</code>	<code>"\$100.00"</code>	Floating \$ sign and zero.
<code>PHONE=5136122300</code> <code>FORMAT(PHONE, "(###) ###-####")</code>	<code>"(513) 612-2300"</code>	Formatting and punctuation.
<code>FORMAT(PART_NO, "PART #????", "?")</code>	<code>"PART # 38765"</code>	Uses literals (PART) and allows # to be used as literal text. "?" is the digit-select character.
<code>FORMAT(ZERO, "#####Z")</code>	<code>"0"</code>	Single zero forced.
<code>FORMAT(123, "Z#####")</code>	<code>"000123"</code>	Leading zeros forced.

- ◆ The following example shows that the FORMAT function can provide formatted output for SHOW fields or fields on a screen or file that are not normally formatted:

```
00010  SHOW NAME,AT(40), FORMAT(ACCT_NO,"Z##-####-#"), AT(55),
00020  .  FORMAT(BALANCE, "##,##Z.##CR")
```

Produces the following result:

```
HENRIETTA JOHNSON          034-4783-1      127.89CR
```

# FSI

The FSI function indicates the success or failure of a logical view, MANTIS file, or external file GET, DELETE, INSERT, RELEASE, or UPDATE.

**FSI(*name*[,*msg*])**

## *name*

<b>Description</b>	<i>Required.</i> Specifies the name for the logical view, MANTIS file, or an external file view.
<b>Format</b>	A MANTIS symbolic name (see “ <a href="#">Symbolic names</a> ” on page 24)

## *msg*

<b>Description</b>	<i>Optional.</i> Specifies the symbolic name you supply for the External file DELETE, GET, INSERT, or UPDATE function message.
<b>Format</b>	A MANTIS symbolic name, specified as a text variable, with a length of at least 40 (see “ <a href="#">Symbolic names</a> ” on page 24)

## General considerations

- ◆ MANTIS file and external file DELETE generic ALL returns the number of records deleted in the message area of the FSI in the format XXXXX DELETED RECORDS. All asterisks in the XXXXX area indicate a record number greater than five characters.
- ◆ Function Status Indicators (FSIs) reflect the success or failure of your command. See “[Status functions](#)” on page 517 for more details on FSIs, logical views, and external files.
- ◆ See also “[ASI](#)” on page 93 and “[VSI](#)” on page 505.



## Examples

- ◆ The following example shows the basic use of the FSI function:

```
00020 VIEW CUSTOMERS( "NEW_CUSTOMERS" )
.
.
.
00110 GET CUSTOMERS
00120 IF FSI(CUSTOMERS) <> "GOOD"
.
.
```

- ◆ The following example shows how to use the FSI function to test for a specific value and have MANTIS return an error message:

```
00010 ACCESS X ( "TEXT", "PSW" )
00020 TEXT MESSAGE( 40 )
00030 TRAP X ON
00040 GET X FIRST
00050 IF X="ERROR"
00060 .IF FSI(X,MESSAGE)="UNAVAILABLE"
00070 ..SHOW "FILE TEST IS UNAVAILABLE"
00080 .ELSE
00090 ..SHOW "FILE TEST GET FAILED:STATUS="+MESSAGE
00100 .END
00105 .WAIT
00120 END
```

- ◆ The following example shows the debugging use of FSI:

```
COMMAND ==> SHOW FSI(X,MESSAGE), MESSAGE
```

# GET

The GET statement reads a record from an external file, a MANTIS file, a personal computer file, an RDM logical view, or a TOTAL DBMS view. Before you can read from a file or view, you must open it by processing the associated FILE, TOTAL, ACCESS, or VIEW statement.

## GET (External file)

```
GET file - name [(key1, key2, . . . )EQUAL] [ENQUEUE] [LEVEL = n]
                FIRST
                NEXT
                PRIOR
                LAST
```

**file-name**

- Description**     *Required.* Specifies the name (as defined in a previously executed ACCESS statement) of the file you want to access.
- Format**            A MANTIS symbolic name (see “Symbolic names” on page 24)

---

**key1,key2,...**

**Description**     *Optional.* Specifies the record key(s) of the desired record. MANTIS assigns values to the corresponding key elements in the file profile (that is, *key1* is the first key element, *key2* is the second, and so on). For external files, if you omit this parameter, MANTIS retrieves the next record.

**Format**            Text, numeric, or DBCS expression

**Considerations**

- ◆ Not all keys need to be specified.
- ◆ For SEQUENTIAL files, the key you supply specifies the Relative Byte Address (RBA) of the record to be retrieved. The first record has an RBA of 0, and each subsequent record has an RBA value of the previous record plus its length.
- ◆ For NUMBERED files, the key is the Relative Record Number (RRN). The first record has an RRN of 1.
- ◆ The order of the specified keys must correspond to the order of key declarations in the file. You cannot omit a key that occurs before or between keys you want to specify. For example, you cannot specify *key1* and *key3* without specifying *key2*, or specify *key3* without specifying both *key1* and *key2*.

---

**EQUAL**

**Description**     *Optional.* Tells MANTIS to retrieve only an exact key match. MANTIS returns “NOTFOUND” if it cannot find the identical key.

**FIRST/NEXT/PRIOR/LAST**

<b>Description</b>	<i>Optional.</i> Specifies the location of the logical record that is to be deleted relative to the current positioning.
<b>Default</b>	NEXT
<b>Format</b>	Must be coded exactly as shown
<b>Options</b>	<p>FIRST Retrieves a record at the beginning of an external file in a sequential retrieval mode.</p> <p>NEXT Retrieves the subsequent file record in a keyless retrieval mode.</p> <p>PRIOR Retrieves the previous sequential record in sequential retrieval mode. If no position exists in a file, the last record is returned with a return status of "NEXT".</p> <p>LAST Retrieves the last record in an external file in a sequential retrieval mode.</p>

**Considerations**

- ◆ For SEQUENTIAL files, a GET FIRST is required to return the first record in the file.
- ◆ GET PRIOR and GET LAST are not supported in the IMS environment.

---

## ENQUEUE

**Description**     *Optional.* Retrieves the record and holds it where it can be updated or deleted without another task having access to it.

### Considerations

- ◆ Only one ENQUEUEd record per external file is permitted at one time. If you have a LEVEEd file, only the last record retrieved is ENQUEUEEd.
- ◆ ENQUEUEs are released on the dequeue that occurs in a subsequent GET *file-name*, interface call, DEQUEUE *file-name*, COMMIT, or Terminal I/O (with COMMIT ON).
- ◆ GET *record* ENQUEUE operation locks out all other users' modifications to the record being retrieved until the record has been dequeued.
- ◆ ENQUEUEing on a record prevents any other application from updating the record; therefore, if you choose not to update or delete the record, dequeue the record as soon as possible. Failure to do so may degrade performance because other applications wait for access to the record.
- ◆ Issue the ENQUEUE option only when you probably will update/delete the record. Otherwise, you may degrade overall performance by serializing resources. However, using the GET ... ENQUEUE will save a second READ if you UPDATE or DELETE the record.
- ◆ If you UPDATE or DELETE a record obtained via a GET ... ENQUEUE, that record is held until the end of the logical unit of work.
- ◆ The GET with ENQUEUE option checks the file's UPDATE password to determine if an update is allowed; if not, a GET without ENQUEUE is issued. Any attempt to update or delete after that will return an error message or a "LOCK" status.

- ◆ ENQUEUE can be specified when running MANTIS in batch, but has no meaning in this mode.
- ◆ If you have multiple ACCESS statements describing the same external VSAM file, only the last GET with ENQUEUE (regardless of which ACCESS statement it pertains to) is enqueued. Incorrect use of the GET with ENQUEUE, with multiple ACCESS statements defining the same external, can result in errors on subsequent UPDATE or DELETE statements in some environments. For example, the following program stops and issues an error when run under CICS:

```
00030 .ENTRY ENQUEUE_PROGRAM
00040 .ACCESS F1("USER:FILE1", "PWD")
00050 ..ACCESS F2("USER:FILE2", "PWD")
00060 .GET F1("KEY1") EQUAL ENQUEUE
00070 .GET F2("KEY2") EQUAL ENQUEUE
00080 .UPDATE F1
00090 .UPDATE F2
00100 .COMMIT
00110 .EXIT
```

The FILE1 and FILE2 external file views both reference the same physical data set. MANTIS issues an error when processing the “UPDATE F1” statement (line 80) under CICS, because CICS returns an error when MANTIS attempts an internal get-for-update against F1. CICS allows only one get-for-update operation per data set at a time, and an outstanding get-for-update request is already in effect for the data set from the “GET F2 . . ENQUEUE” statement.

---

**LEVEL=*n***

<b>Description</b>	<i>Optional.</i> Specifies the number of the buffer that contains the record you want to get.
<b>Default</b>	1
<b>Format</b>	Arithmetic expression that evaluates to a value in the range of 1 through <i>m</i> , where <i>m</i> is the maximum buffer number, as defined in the corresponding ACCESS statement

**Considerations**

- ◆ Only specify LEVEL=*n* when the *file-name* has buffers defined.
- ◆ MANTIS uses only the integer portion of *n*.

**General considerations**

- ◆ An external file open (when required) is issued on the first DELETE, GET, INSERT, or UPDATE to the file.
- ◆ For INDEXED files, the key you supply in the GET statement must correspond partially or completely to the key you specify during the file view design.
- ◆ For SEQUENTIAL files, MANTIS returns the RBA of the retrieved record in the associated reference variable.
- ◆ For NUMBERED files, MANTIS returns the RRN of the retrieved record in the associated reference variable.

- ◆ MANTIS returns a text string in the variable called *file-name* that reflects the status of the operation:

Returned text string	Description
"FOUND"	MANTIS successfully retrieved the record identified by the supplied key.
"END"	MANTIS did not retrieve the record because it reached the end of the file using the NEXT option or the beginning of the file using the PRIOR option.
"NEXT"	When you use a partial key for a generic search, MANTIS returns a status of "NEXT" in the <i>file-name</i> unless it reaches the end of file (and a status of "END"). MANTIS retrieved the next record in a sequential GET statement (without a key); or MANTIS retrieved the previous sequential record because you issued the GET statement with PRIOR option; or MANTIS retrieved the last record in a file using the LAST option (or the FIRST option returned the first record). MANTIS also returns this value where it could not locate the actual key, and the system returned the next record in the sequence.
"NOTFOUND"	<p>You requested an exact key match with the "EQUAL" option, but MANTIS could not find an exact match. MANTIS doesn't change program variables for elements in the file.</p> <p>A GET EQUAL against a nonunique alternate key index returns a NOTFOUND if the key is in fact a duplicate.</p>
"LOCK" *	The password specified in the ACCESS statement for this file view is not valid.



Returned text string	Description
"ERROR" *	<p>MANTIS received an error status. Use the FSI function (see "FSI" on page 232) for more information. Possible causes:</p> <ul style="list-style-type: none"> <li>◆ A physical error occurred during record retrieval.</li> <li>◆ RBA specified for a GET to SEQUENTIAL files was not at a record boundary.</li> <li>◆ RRN for NUMBERED files specified a record number outside the file range.</li> <li>◆ The file view definition does not correctly reflect the physical file you are accessing.</li> <li>◆ The External file exit canceled the operation.</li> </ul>
"DATA" *	A field that should be numeric is not.
"NOTOPEN" *	The external file is not open.
"NOTSUPP" *	You requested an operation that is not supported by the TP monitor.

\* Returned only when TRAP is in effect for the file.

- ◆ The following table clarifies the meaning of file status “FOUND”, “NOTFOUND”, for GET options “NEXT”, and “EQUAL” when used with full or partial keys:

Key	Option	Status	FSI	Comments
Full	EQUAL	FOUND	GOOD	Exact match. Record retrieved.
Full	EQUAL	NOTFOUND	NOTFOUND	No match. Record not retrieved.
Partial	EQUAL	NOTFOUND	NOTFOUND	Will never match. Record was not retrieved. Do not use EQUAL with a partial key.
Full	NEXT	FOUND	GOOD	Exact match. Record retrieved.
Full	NEXT	NEXT	NOTFOUND	Close match. Record retrieved, but key is higher than supplied key.
Full	NEXT	END	NOTFOUND	No match. Record not retrieved. Key supplied is higher than any key in the file.
Partial	NEXT	NEXT	NOTFOUND	Exact or close match. Record retrieved. May be an exact match for the supplied key or may be next record after supplied key. Compare the supplied key to the returned key to determine which is the case.
Partial	NEXT	END	NOTFOUND	No match. Record not retrieved. Key supplied is higher than any key in the file.

- ◆ If you issue GET EQUAL, a subsequent GET without keys, LAST, or FIRST returns the next sequential record. This occurs even if MANTIS returns “NOTFOUND” on the GET EQUAL.
- ◆ Sequential GETs (FIRST, LAST, NEXT, or PRIOR) issued against an empty file returns “END”.
- ◆ For extended external file status messages and Function Status Indicators (FSIs), see “[Extended status messages for MANTIS and external files](#)” on page 521.

- ◆ When more than one file needs to be ENQUEUEd, the application program must ensure that the sequence of the enqueued files is always the same and is identical to the sequence used by other programs doing ENQUEUEs. If this rule is not observed, application program lockouts can occur due to two or more programs trying to obtain exclusive control of resources that are already controlled by these programs.
- ◆ The External File Exit can affect this statement. See your Master User for details.
- ◆ See also “DELETE” on page 183, “DEQUEUE” on page 203, “FSI” on page 232, the ENQUEUE parameter under the “GET” statements (the “GET” statements start on page 234), “INSERT” on page 277, “TRAP” on page 469, and “UPDATE” on page 479.

### Example

The following example shows how to code an EXTERNAL FILE GET. In this environment, the file is opened on the first GET statement.

```

00020 .ACCESS RECORD( "INDEX", "SERENDIPITY",16)
00030 .SCREEN MAP( "INDEX" )
00040 .CONVERSE MAP
00050 .COUNTER=1
00060 .WHILE MAP<>"CANCEL" AND COUNTER<17
00070 ..WHEN INDICATOR(COUNTER)="G"
00080 ...GET RECORD LEVEL=COUNTER
00090 ..WHEN INDICATOR(COUNTER)="D"
00100 ...DELETE RECORD LEVEL=COUNTER
.
.
.
```

## **MANTIS external VSAM KSDS nonunique alternate key processing**

### **The problem of skipping records**

A few basic rules MANTIS follows:

- ◆ Records are accessed directly (GET filename(key)EQUAL) or sequentially (GET filename(key), GET FIRST, GET NEXT, GET PRIOR, GET LAST). In CICS, READ is used for direct access and STARTBR, RESETBR, READNEXT and READPREV are used for sequential access.
- ◆ When accessed sequentially, if a valid browse pointer does not exist, a browse pointer is established using a START BROWSE command. Direct access does not start a browse. The browse is terminated under the following conditions:
  - CALL
  - COMMIT
  - CONVERSE
  - DELETE
  - EXIT of dolevel where ACCESS defined
  - FAULT (error message)
  - GET key EQUAL
  - INSERT
  - I/O to the file from another ACCESS (string stealing)
  - OBTAIN
  - PERFORM
  - PROMPT
  - RESET
  - SHOW when terminal I/O is forced
  - UPDATE
  - WAIT

- ◆ MANTIS saves the key of the record after a successful direct or sequential GET. This key is used in subsequent I/O where a key is not specified.
- ◆ After a browse has been terminated, on a subsequent GET NEXT, MANTIS reestablishes the browse pointer by adding a binary one to the key saved from the last successful GET and issuing a START BROWSE.

With the previous rules in mind, MANTIS' processing of nonunique alternate keys can be understood. The following examples are oversimplified and do not contain all error checking needed for a good program:

### Example 1

```

10 ACCESS FILE1(...)
20 GET FILE1
30 WHILE FILE1 <> "END"
40 .SHOW SEC_KEY:WAIT
50 .GET FILE1
60 END

```

In this example, the key field for an ACCESS defined for an alternate index is being displayed after each GET NEXT. In statement 20, a START BROWSE and a READ NEXT are issued. If the read is successful, the key is saved. In statement 40, the terminal I/O terminates the browse. In statement 50, another START BROWSE is issued using a key equal to one more than the key saved from the last successful GET. If this program is run against a file with nonunique alternate keys, only the first of each set of equal keys is displayed due to the repositioning of the lost browse pointer. Removing the WAIT from statement 40 and adding a WAIT at the end of this program nearly solves this problem. After enough SHOWs have been issued to fill the screen, MANTIS does an implied WAIT that terminates the browse. If this implied WAIT occurs in the middle of a set of equal keys, the remaining equal keys will be skipped when MANTIS reestablishes the browse pointer.

To solve this problem, the application must take into account when the browse pointer may be lost due to a CONVERSE, COMMIT, and so on. One solution is to save the key value and maintain a counter that is reset at the beginning of each new set of equal keys. After the browse pointer is lost, the application uses the key and counter to reposition on the correct record by issuing a GET filename(key) and a number of GET NEXTs equal to the counter value.

An alternate solution is to read all of the records having the same key into an array before issuing the MANTIS statement that loses the browse pointer. Then the record data may be displayed from the array. This second solution may not be practical in some cases. There may be too much data involved. Or updates may have been done to these records by another transaction during your terminal I/O.

### Example 2

```
10 ACCESS FILE1(...)  
20 GET FILE1(key)EQUAL  
30 WHILE FILE1 <> "END"  
40 .SHOW SEC_KEY  
50 .GET FILE1  
60 END  
70 WAIT
```

This example is similar to the first, except that line 20 has been changed to a GET EQUAL and the WAIT has been placed at the end of the program.

Assume:

- ◆ The screen has an infinite number of rows so that we don't have to be concerned with losing the browse pointer due to an implied WAIT.
- ◆ The alternate key values in the first and second records match the key specified in statement 20. (See the following table.)
- ◆ The alternate key value in the third record does not match the key specified in statement 20.

In statement 20, the GET EQUAL reads the first record successfully and saves the key, but does not start a browse. The first time statement 50 is executed, MANTIS skips the second record and reads the third record. This is because a browse has not been started. Consequently MANTIS adds a binary one to the key saved in statement 20 and starts a browse with that key. If statement 20 is changed to "GET FILE1(key)" the problem of skipping records is solved because statement 20 will now start the browse.

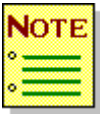
### **A comparison of GET filename(key) and GET filename(key)EQUAL**

Assume the following alternate key values:

<b>Record no.</b>	<b>Alt. key Value</b>
1	A
2	A
3	C
4	C
5	D

The following table summarizes what is returned for various statements:

Statement	FILE1	FSI(FILE1, MESSAGE)	MESSAGE	Data for record
GET FILE1("A")	FOUND	GOOD	" "	1
GET FILE1("A")EQUAL	FOUND	GOOD	filename DUPLICATE KEY	1
GET FILE1("B")	NEXT	NOTFOUND	" "	3
GET FILE1("B")EQUAL	NOTFOUND	NOTFOUND	" "	None
GET FILE1("D")EQUAL	FOUND	GOOD	" "	5



GET FILE1("B")EQUAL returns a file status of NOTFOUND, but still saves key "B". A subsequent GET FILE1 NEXT returns record 3.



# GET (MANTIS file)

```

GET file - name [ (key1, key2, . . . ) [EQUAL] ] [ENQUEUE] [LEVEL = n]
                  FIRST
                  NEXT
                  PRIOR
                  LAST
    
```

## file-name

<b>Description</b>	<i>Required.</i> Specifies the name (as defined in a previously executed FILE statement) of the file you want to access.
<b>Format</b>	A MANTIS symbolic name (see “ <a href="#">Symbolic names</a> ” on page 24)

## key1,key2,...

<b>Description</b>	<i>Optional.</i> Specifies the record key(s) of the desired record. MANTIS assigns values to the corresponding key elements in the file profile (that is, <i>key1</i> is the first key element <i>key2</i> is the second, and so on).
<b>Format</b>	An arithmetic, text, or DBCS expression
<b>Considerations</b>	<ul style="list-style-type: none"> <li>◆ Not all keys need to be specified.</li> <li>◆ The order of the specified keys must correspond to the order of key declarations in the file. You cannot omit a key that occurs before or between keys that you want to specify. For example, you cannot specify <i>key1</i> and <i>key3</i> without specifying <i>key2</i>, or specify <i>key3</i> without specifying both <i>key1</i> and <i>key2</i>.</li> </ul>

## EQUAL

<b>Description</b>	<i>Optional.</i> Tells MANTIS to retrieve only an exact key match. MANTIS returns “NOTFOUND” if it cannot find the identical key.
<b>Consideration</b>	If you issue GET EQUAL, a subsequent GET without keys, LAST, or FIRST returns the next sequential record. This occurs even if MANTIS returns “NOTFOUND” on the GET EQUAL.

---

**FIRST/NEXT/PRIOR/LAST**

<b>Description</b>	<i>Optional.</i> Specifies the location of the logical record that is to be deleted relative to the current positioning.
<b>Default</b>	NEXT
<b>Format</b>	Must be coded exactly as shown
<b>Options</b>	<p>FIRST Retrieves a record at the beginning of a MANTIS file in a sequential retrieval mode.</p> <p>NEXT Retrieves the subsequent file record in a keyless retrieval mode.</p> <p>PRIOR Retrieves the previous sequential record in sequential retrieval mode. If no position exists in a file, the last record is returned with a return status of "NEXT".</p> <p>LAST Retrieves the last record in a file in a sequential retrieval mode.</p>

**Considerations**

- ◆ GET FILE FIRST retrieves the first key on the file. GET FILE (key) starts reading records at the key specified position. GET FILE (key) EQUAL retrieves a specific record. GET FILE retrieves the next record on the file.
- ◆ GET PRIOR and GET LAST are not supported in the IMS environment, or by Entity Transformers.

---

**ENQUEUE**

<b>Description</b>	<i>Optional.</i> This parameter is for prototyping purposes only because it is not valid for MANTIS internal files and is ignored if specified. ENQUEUE is useful if you plan to convert the file later to an EXTERNAL FILE and want to use ENQUEUE capabilities there.
--------------------	---

---

**LEVEL=*n***

**Description**     *Optional.* Specifies the number of the buffer containing the record to get.

**Default**            1

**Format**            Arithmetic expression that evaluates a value in the range of 1 through *m*, where *m* is the maximum buffer number, as defined in the corresponding FILE statement

**Considerations**

- ◆ Only specify LEVEL=*n* when the *file-name* has buffers defined.
- ◆ MANTIS uses only the integer portion of *n*.

## General considerations

- ◆ MANTIS returns a text string in the variable named *file-name* that reflects the status of the operation:

Returned text string	Description
"FOUND"	MANTIS successfully retrieved the record identified by the key you supplied.
"NEXT"	When you use a partial key for a generic search, MANTIS returns a status of "NEXT" in the <i>file-name</i> unless it reaches the end of file (and a status of "END"). NEXT indicates that MANTIS retrieved the next record in a sequential GET statement (without a key); or MANTIS retrieved the previous sequential record because you issued the GET statement with PRIOR option; or MANTIS retrieved the last record in a file using the LAST option (or the FIRST option returned the first record). MANTIS also returns this value where it could not locate the actual key, and the system returned the next record in the sequence.
"END"	MANTIS failed to retrieve the record because it reached the end of the file using the NEXT option or the beginning of the file using the PRIOR option.
"NOTFOUND"	MANTIS could not find the record you specified using the key and EQUAL options.
"ERROR" *	MANTIS received an error status. Use the FSI function (see "FSI" on page 232) for more information. Possible causes: <ul style="list-style-type: none"><li>◆ A physical error occurred during record retrieval.</li><li>◆ The SETPRAY Exit canceled the operation.</li></ul>
"LOCK" *	The password specified in the FILE statement is invalid.

\* Returned only when TRAP is in effect for the file.

- ◆ For extended MANTIS file status messages and Function Status Indicators (FSIs), see “[Extended status messages for MANTIS and external files](#)” on page 521.
- ◆ Sequential GETs (FIRST, LAST, NEXT, or PRIOR) issued against an empty file returns “END”.
- ◆ ENQUEUEing on MANTIS internal files is ignored.
- ◆ The Setpray Exit can affect this statement. See your Master User for details.
- ◆ See also “[DELETE](#)” on page 183, “[FSI](#)” on page 232, “[INSERT](#)” on page 277, “[TRAP](#)” on page 469, and “[UPDATE](#)” on page 479.

### Example

The following example shows how a MANTIS file is accessed via GETs in keyed retrieval mode and sequential retrieval mode:

```

00020 FILE RECORD( "INDEX", "SERENDIPITY", 16 )
00030 SCREEN MAP( "INDEX" )
00040 WHILE RECORD<>"END" AND MAP<>"CANCEL"
00050 .CLEAR MAP:BUFFER=1
00070 .GET RECORD( "WILLIAMS" ) LEVEL=BUFFER
00080 .WHILE RECORD<>"END" AND BUFFER<17
00090 ..BUFFER=BUFFER+1
00100 ..GET RECORD LEVEL=BUFFER
00110 .END
00120 .CONVERSE MAP
00130 END

```

## GET (Personal computer file)

GET file - name

(key)[EQUAL]

FIRST

NEXT

PRIOR

LAST

[LEVEL = n]

### file-name

- Description

Required. Specifies the name (as defined in a previously executed ACCESS statement) of the file you want to access.
- Format

A MANTIS symbolic name (see “Symbolic names” on page 24)

### key

- Description

Optional. Specifies the key(s) of the desired record. MANTIS assigns values to the corresponding key element in the file profile.
- Consideration

For SEQUENTIAL files, the key is the Relative Byte Address (RBA). For NUMBERED files, the key is the Relative Record Number (RRN). The first record had an RRN of 1.

### EQUAL

- Description

Optional. Tells MANTIS to retrieve only an exact key match.
- Consideration

MANTIS returns “NOTFOUND” if it cannot find the identical key.

### FIRST/NEXT/PRIOR/LAST

- Description

Optional. Specifies the location of the logical record that is to be deleted relative to the current positioning.
- Default

NEXT
- Options

FIRST

Retrieves a record at the beginning of a file in a keyless retrieval mode.

NEXT

Retrieves the subsequent record in a file.

PRIOR

Retrieves the previous record in a file.

LAST

Retrieves the last record in a file.

---

**LEVEL=*n***

<b>Description</b>	<i>Optional.</i> Specifies the number of the buffer that contains the record you want to get.
<b>Default</b>	1
<b>Format</b>	Arithmetic expression that evaluates to a value in the range of 1 through <i>m</i> , where <i>m</i> is the maximum buffer number, as defined in the corresponding ACCESS statement

**Consideration**

- ◆ Only specify LEVEL=*n* when the *file-name* has buffers defined.
- ◆ MANTIS uses only the integer portion of *n*.

General considerations

- ◆ MANTIS returns a text string in the variable named *file-name* that reflects the status of the operation.

Returned text string	Description
"FOUND"	MANTIS successfully retrieved the record identified by the key you supplied.
"NEXT"	MANTIS retrieved the next record in sequence because you issued the GET statement without a key. MANTIS also returns this value for a GET FIRST statement or for a GET PRIOR statement.
"END"	MANTIS failed to retrieve the record for a GET NEXT statement because it reached the end of the file. MANTIS also returns this value for a GET PRIOR statement when it reaches the beginning of the file.
"NOTFOUND"	MANTIS returns "NOTFOUND" when it cannot find a record you specify using the key and EQUAL options.
"LOCK" *	The password specified in the ACCESS statement is invalid.
"ERROR" *	MANTIS received an error status. Use the FSI function (see "FSI" on page 232) for more information.

\* Returned only when TRAP is in effect for the file.



For extended personal computer file status messages and Function Status Indicators (FSIs), see “[Extended status messages for MANTIS and external files](#)” on page 521.

- ◆ For NUMBERED files, MANTIS returns the RRN of the retrieved record in the associated reference variable.
- ◆ For SEQUENTIAL files, MANTIS returns the RBA of the retrieved record.
- ◆ For NUMBERED PC files, if a NUMBERED record consists entirely of binary zeros, PC CONTACT assumes the record does not exist and does not return a value for the record.
- ◆ Files residing on personal computer files can only be accessed by that personal computer user.
- ◆ For PC files, MANTIS uses buffering for GET NEXTs and INSERTs to increase the speed of uploads and downloads. A common upload program issues many GET NEXTs in succession against a PC file, writing the records to a mainframe file. On the first GET NEXT, MANTIS receives only the number of records that fit in the buffer, beginning with the record expected for that GET NEXT. For subsequent GET NEXTs, those records are usually already in the buffer, eliminating unnecessary I/O to the PC. (This buffering is independent of LEVEL usage, which refers to MANTIS variables buffering.)

- ◆ To take full advantage of GET NEXT buffering, avoid the following events that cause an INSERT buffer to be sent to the PC or a GET NEXT buffer to be discarded:
  - Task termination
  - Program termination
  - EXIT from an externally DOne program containing the ACCESS statement for the PC file (a special case of program termination)
  - COMMIT
  - Screen I/O
  - Perform of an external interface
  - Change of I/O operation such as changing from a GET NEXT to a GET (by key), GET PRIOR, GET LAST, UPDATE, INSERT, or DELETE
  - Full buffer (for INSERTs)
- ◆ All GET NEXT errors are trappable using the TRAP statement because the error is detected at the time the MANTIS program issues the GET NEXT.
- ◆ See also “DELETE” on page 183, “FSI” on page 232, “INSERT” on page 277, “TRAP” on page 469, and “UPDATE” on page 479.

### Example

```
00010 ENTRY PC_EXAMPLE
00020 .VIEW V_REC( "SAMPLE_VIEW" )
00030 .ACCESS PC_REC( "SAMPLE_PC_ONE" , "REMOVE" )
00040 .GET PC_REC
00050 .WHILE PC_REC<>"END"
00060 ..INSERT V_REC
00070 .GET PC_REC
00080 .END
00090 .SHOW "FILE TRANSFER COMPLETE":WAIT
00100 EXIT
```

## GET (RDM logical view)

**GET** *view - name*  $\left[ \begin{array}{l} (key1, key2, \dots) \\ \text{AT mark - name} \\ \text{SAME} \end{array} \right] \left[ \begin{array}{l} \text{NEXT} \\ \text{PRIOR} \\ \text{FIRST} \\ \text{LAST} \end{array} \right] [\text{ENQUEUE}][\text{LEVEL} = n]$

***view-name***

<b>Description</b>	<i>Required.</i> Specifies the name (as defined in a previously executed VIEW statement) of the logical view you want to access.
<b>Format</b>	A MANTIS symbolic name (see “ <b>Symbolic names</b> ” on page 24)

**key1,key2,...**

<b>Description</b>	<i>Optional.</i> Specifies the record key(s) of the desired record. MANTIS assigns values to the corresponding key elements in the view profile ( <i>key1</i> is the first key element, <i>key2</i> is the second, and so on).
--------------------	--

## Considerations

- ◆ Not all keys need to be specified.
- ◆ MANTIS treats any omitted keys as generic keys. Using generic keys allows direct access to a specific record or sequential access to many records. All occurrences of a particular unspecified key are returned as long as the other keys are satisfied.
- ◆ The order of the specified keys must correspond to the order of key declarations in the logical view. You cannot omit a key that occurs before or between keys that you want to specify. For example, you cannot specify *key1* and *key3* without specifying *key2*, or specify *key3* without specifying both *key1* and *key2*.
- ◆ If you don't need to be certain of the content of the logical record, you can use a GET without the ENQUEUE option. When you execute the UPDATE or DELETE statements, the automatic hold facility performs the lock prior to modifying the record.

---

## AT mark

- Description** *Optional.* Repositions to the logical view previously marked with the MARK statement.
- Format** 4-character text expression
- Consideration** The mark-expression must be set by a previously executed MARK statement.

---

## SAME

- Description** *Optional.* Retrieves the same logical view record previously accessed. If no current record exists, MANTIS returns a “NOTFOUND” status.
- Format** Must be coded exactly as shown
- Consideration** If there is only one logical record for a given key and you try to use the same key with a keyed GET statement to access the logical record a second time, you receive a “NOTFOUND” status. This message indicates that there are no more occurrences of this particular logical record. In order to access this same logical record, use a GET SAME statement instead.

---

**FIRST/NEXT/PRIOR/LAST**

<b>Description</b>	<i>Optional.</i> Specifies the location of the logical record that is to be read relative to the current positioning.
<b>Default</b>	NEXT
<b>Options</b>	<p><b>FIRST</b> Retrieves the first logical record in the logical view with the specified keys. If you don't supply keys, MANTIS returns the first record.</p> <p><b>NEXT</b> Retrieves the next record in the logical view with the specified keys. If you don't supply keys, MANTIS returns the next sequential record. If no current record exists, GET NEXT operates as GET LAST.</p> <p><b>PRIOR</b> Retrieves the previous record with the specified keys. If no current record exists, GET PRIOR operates as GET FIRST. If you don't supply keys, MANTIS retrieves the record before the currently established position of a given key. However, after processing all prior records for a key, MANTIS displays "DBMS DOES NOT SUPPORT THIS OPERATION" and halts execution. If TRAP is in effect, MANTIS returns "ERROR".</p> <p><b>LAST</b> Retrieves the last logical view record in the logical view for the specified keys. If you don't supply keys, MANTIS returns the last record.</p>

**Considerations**

- ◆ If the underlying file system cannot perform the GET PRIOR or GET LAST functions, MANTIS displays a message and halts execution. If TRAP is in effect, MANTIS returns the "ERROR" status.
- ◆ A series of GET NEXTs loops back to the first record and continues if you don't check for a "NOTFOUND" status.

## ENQUEUE

---

**Description**     *Optional.* Locks out all other users' modifications to the logical view record being retrieved until the record has been updated, deleted, or dequeued by a COMMIT statement.

**Format**            Must be coded exactly as shown

### Considerations

- ◆ If you don't need to be certain of the content of the logical record, you can use a GET without the ENQUEUE option. When you execute the UPDATE or DELETE statements, the automatic hold facility performs the lock prior to modifying the record.
- ◆ If you use an ENQUEUE on any GET statement since the last terminal I/O, MANTIS automatically issues a COMMIT prior to any further terminal I/O.

**LEVEL=*n***

<b>Description</b>	<i>Optional.</i> Specifies the number of the buffer that contains the record you want to get.
<b>Default</b>	1
<b>Format</b>	Arithmetic expression that evaluates to a value in the range of 1 through <i>m</i> , where <i>m</i> is the maximum buffer number, as defined in the corresponding VIEW statement

**Consideration**

- ◆ Only specify LEVEL=*n* when the *file-name* has buffers defined.
- ◆ MANTIS uses only the integer portion of *n*.

**General considerations**

- ◆ You must execute a corresponding VIEW statement before you can execute the GET statement.
- ◆ MANTIS returns a text string in the variable name *view-name* that reflects the status of the operation.

Returned text string	Description
"FOUND"	MANTIS successfully retrieved the record identified by the supplied key(s).
"NOTFOUND"	The record identified by the supplied key(s) doesn't exist.
"DATA"	A field that should be numeric is not.
"LOCK" *	The password specified in the VIEW statement for this view is not valid.
"ERROR" *	<p>MANTIS received an error status. Use the FSI function (see "FSI" on page 232) for more information. Possible causes:</p> <ul style="list-style-type: none"> <li>◆ An RDM error occurred during database access.</li> <li>◆ You tried to perform an invalid function on the user view.</li> </ul>

\* Returned only when TRAP is in effect for the file.

- ◆ RDM logical view GET sets three status functions to the application program that indicate processing results—FSI, ASI, and VSI. FSI indicates the success or failure of your command. ASI indicates the status of each field in the logical record. VSI indicates the highest field status within the logical record. For a complete discussion of these status functions, see “[Status functions](#)” on page 517.
- ◆ See also “[ASI](#)” on page 93, “[DELETE](#)” on page 183, “[FSI](#)” on page 232, “[INSERT](#)” on page 277, “[TRAP](#)” on page 469, “[UPDATE](#)” on page 479, and “[VSI](#)” on page 505.

**Example**

The following example shows how an RDM LOGICAL VIEW GET is coded:

```
00010 VIEW CUSTOMER( "CUST" )
00020 SCREEN MAP( "CUST_UPDATE" )
00050 GET CUSTOMER FIRST
00060 WHILE CUSTOMER="FOUND" AND MAP<>"CANCEL"
00090 .CONVERSE MAP
00100 .GET CUSTOMER
00140 END
```



GET (TOTAL file view)

GET <i>file - name</i>	SET ( <i>key1, key2, ...</i> )	AT refer	[ENQUEUE][LEVEL = <i>n</i> ]
		BEFORE refer	
		AFTER refer	
		FIRST	
		LAST	
	( <i>key1, key2, ...</i> )		
	[FIRST]		

file-name

**Description**     *Required.* Specifies the name (as defined in a previously executed TOTAL statement) of the TOTAL view you want to access.

**Format**             A MANTIS symbolic name (see “Symbolic names” on page 24)

SET(key1, key2, ...)

**Description**     *Optional.* A SET is a chain of variable-entry records. Records belonging to the SET do not have individual keys you can use to reference them. Instead, a key identifies the whole set. Once the SET is identified, you can retrieve individual records in a sequential manner, normally starting at the top and going down.

**Format**             *keyn* Must be a text, numeric, or DBCS expression that matches the TOTAL definition in type, size, and other dimensions and is coded exactly as shown

Considerations

- ◆ Applies to variable-entry files only.
- ◆ When you use the SET parameter, MANTIS automatically updates the *refer* variable.

---

**AT/BEFORE/AFTER/FIRST/LAST**

<b>Description</b>	<i>Optional.</i> Specifies the location of the logical record that is to be deleted relative to the current positioning.
<b>Default</b>	AFTER <i>refer</i>
<b>Options</b>	AT <i>refer</i> Retrieves a specific record in the set.  BEFORE <i>refer</i> Retrieves the record prior to the one specified by <i>refer</i> .  AFTER <i>refer</i> Retrieves the record after the one specified by <i>refer</i> .  FIRST Retrieves the record at the beginning of the set.  LAST Retrieves the record at the end of the set.

---

**refer**

<b>Description</b>	<i>Optional.</i> Identifies the position in the database file where you can locate the record.
<b>Format</b>	4-character text expression
<b>Consideration</b>	You can save and restore a <i>refer</i> using a text variable of at least four characters in length. The Reference Variable name found on the TOTAL File View Design Facility menu holds the <i>refer</i> value. For more details, refer to <i>MANTIS Facilities, OS/390, VSE/ESA</i> , P39-5001.

---

**key1,key2...**

<b>Description</b>	<i>Optional.</i> Specifies the record key(s) of the desired record. MANTIS assigns values to the corresponding key elements in the TOTAL profile ( <i>key1</i> is the first key element, <i>key2</i> is the second, and so on). For TOTAL, key applies to master files only.
<b>Consideration</b>	Keys must correspond in number, type, and size to the TOTAL definition.

---

**FIRST**

<b>Description</b>	<i>Optional.</i> Points MANTIS to the beginning of a TOTAL file in a sequential retrieval mode.
<b>Format</b>	Must be coded exactly as shown

---

## ENQUEUE

**Description**     *Optional.* Locks out all other users' modifications to the TOTAL DBMS being retrieved until the record has been updated, deleted, or dequeued by a COMMIT statement or terminal I/O.

**Consideration** Must be coded exactly as shown.

---

## LEVEL=*n*

**Description**     *Optional.* Specifies the number of the buffer that contains the record you want to get.

**Default**            1

**Format**            Arithmetic expression that evaluates to a value in the range of 1 through *m*, where *m* is the maximum buffer number, as defined in the corresponding TOTAL statement

### Considerations

- ◆ Only specify LEVEL=*n* when the *file-name* has buffers defined.
- ◆ MANTIS uses only the integer portion of *n*.

## General considerations

- ◆ MANTIS retrieves the record from the database and assigns the data to MANTIS variables, according to the conversion rules defined in your TOTAL view (refer to *MANTIS Facilities, OS/390, VSE/ESA*, P39-5001, for details).
- ◆ A TOTAL file open is issued on the first DELETE, GET, INSERT, or UPDATE.
- ◆ MANTIS returns a text string in a variable named *view-name* that reflects the status of the operation.

Returned text string	Description
"END"	MANTIS failed to retrieve the record because it reached the end of the file.
"NEXT"	MANTIS retrieved the next record in sequence because you issued the GET statement without a key.
"FOUND"	MANTIS successfully retrieved the record identified by the supplied key.
"NOTFOUND"	The master record identified by the supplied key does not exist.
"DATA" *	A field which should be numeric is not.
"LOCK" *	The password specified in the TOTAL statement for this file view is not valid.
"NOTOPEN" *	The TOTAL view is not open.
"NOTAVAL" *	The TOTAL file or view is not open.

\* Returned only when TRAP is in effect for the file.

- ◆ If MANTIS returns a status of "END" after serially reading through a master or variable file or a variable chain, the next READ starts at the beginning of the file or chain.
- ◆ See also "DELETE" on page 183, "INSERT" on page 277, "TRAP" on page 469, and "UPDATE" on page 479.

**Example**

The following example shows how a TOTAL file is processed and data is retrieved using a TOTAL FILE GET:

```
00050 TOTAL CUSTOMERS("CLIENT", "SALES")
00060 TOTAL HISTORY("PAYMENTS", "TEXAS", 11)
00070 TEXT CUSTOMER_ID(20)
00080 SMALL BUFFER
00090 (CUSTOMER_ID)="OUR-BEST"
00100 GET CUSTOMERS(CUSTOMER_ID)
00110 BUFFER=1
00120 GET HISTORY SET(CUSTOMER_ID)FIRST LEVEL=BUFFER
00130 WHILE HISTORY<>"END" AND BUFFER<11
00140 .BUFFER=BUFFER+1
00150 .GET HISTORY SET(CUSTOMER_ID)AFTER
00155 .REFER(BUFFER_1)LEVEL=BUFFER
00160 END
00170 CONVERSE MAP
```

## HEAD

The HEAD statement centers a heading on the top line of the unformatted screen and sets it to high intensity.

---

### HEAD heading

---

---

#### heading

**Description**     *Required.* Specifies the text expression to be displayed at the top of the screen.

**Format**            0–72 character text or DBCS expression

#### General considerations

- ◆ Headings do not appear when you CONVERSE a formatted screen, but only when MANTIS displays unformatted screen output, using an OBTAIN, SHOW, or WAIT.



---

You can also do an OBTAIN and SHOW for formatted screens.

---

- ◆ You can specify only one heading for each screen I/O. If more than one HEAD statement is executed, the most recent HEAD statement specifies the heading for a screen I/O.
- ◆ Specify HEAD "" to clear a previous heading because CLEAR does not affect the heading.
- ◆ See also "SHOW" on page 400.

**Example**

The following example shows how the HEAD statement is used to center a heading on a screen:

```
00010 ENTRY BUZZ_PHRASE_GENERATOR
00020 .DO SET_UP_VOCABULARY
00030 .HEAD"BUZZ PHRASE GENERATOR":SHOW" "
00040 .CLEAR
00050 .SHOW"I WILL GENERATE A SCREEN FULL OF"
00055 .'"BUZZ PHRASES' EVERY"
00060 .'" TIME YOU HIT 'ENTER'. WHEN YOU WANT TO"
00065 .'" STOP, HIT 'PA2'."
00070 .UNTIL KEY="CANCEL"
00080 ..INDEX=1
00090 ..UNTIL INDEX=22
00100 ...A=INT(RND(10)+1)
00110 ...B=INT(RND(10)+1)
00120 ...C=INT(RND(10)+1)
00130 ...SHOW FIRST(A)+" "+SECOND(B)+" "+NOUN(C)
00140 ...INDEX=INDEX+1
00150 ..END
00160 ..WAIT
00170 .END
00180 .CHAIN"GAMES_MENU"
00190 EXIT
```

---

# HELP

The HELP command provides further explanation of an error message, a command, or a list of reserved words. For statements used with END (e.g., WHILE-END, IF-END), do not specify “END” in conjunction with the HELP command (use HELP WHILE or HELP IF).

```
HELP [RESERVED
      command - name
      CODE_xxxxxx
      FSE
      HELP]
```

---

## RESERVED

<b>Description</b>	<i>Optional.</i> Displays a list of all reserved words.
<b>Format</b>	Must be coded exactly as shown

---

## command-name

<b>Description</b>	<i>Optional.</i> Specifies the name of a command or statement that you need explained.
<b>Consideration</b>	Must be a valid MANTIS command or statement as listed in the table under “ <a href="#">MANTIS language summary</a> ” on page 76.

---

## CODE\_xxxxxx

<b>Description</b>	<i>Optional.</i> Specifies the first six characters of the error message code. <a href="#">MANTIS Messages and Codes, OS/390, VSE/ESA</a> , P39-5004 provides more information on error messages.
--------------------	---

---

## FSE

<b>Description</b>	<i>Optional.</i> Displays online help for the Full Screen Editor.
--------------------	---



## HELP

**Description**     *Optional.* Displays online help for the Help Facility.

### General considerations

- ◆ If you are using the Full Screen Editor, the default PF key setting for the HELP command is PF1. Refer to *MANTIS Program Design and Editing, OS/390, VSE/ESA*, P39-5013, for details on changing this assignment.
- ◆ If no HELP exists, you get an error message.

### Examples

The following examples show how to use the HELP function to display online help for various topics:

Example	Comments
HELP	Displays a prompter explaining the last MANTIS message received. If there is no message, a <i>NUCCNHE:Cannot find the specified prompter message</i> is issued.
HELP CONVERSE	Displays a prompter providing a more detailed explanation of the MANTIS command or statement (for example, CONVERSE). For more information, see the table in “ <i>MANTIS language summary</i> ” on page 76. A prompter is available for most of the commands and statements listed there.
HELP RESERVED	Displays all MANTIS reserved words. For a list of MANTIS reserved words, see the table in “ <i>Symbolic names</i> ” on page 24.
HELP CODE_NUCLSI	Displays an explanation of the error that you designate using the six-character ID. An example of the six-character ID is “NUCLSI”.
HELP HELP	Displays a list of commands for which help is available.
HELP FSE	Displays online help information for the Full Screen Editor.
HELP	(In screen design.) Displays a help prompter for PF keys and for Screen Design line commands. For details, refer to <i>MANTIS Facilities, OS/390, VSE/ESA</i> , P39-5001.

---

# IF-ELSE-END

The IF-ELSE-END statement executes a block of statements only if a specified condition (or conditions) is true.

---

```
IF expression
  .blocka
[ ELSE
  .blockb ]
END
```

---

---

## *expression*

- Description**     *Required.* Specifies the condition necessary for MANTIS to execute *blocka*.
- Format**            Relational or arithmetic expression that evaluates to TRUE (nonzero) or FALSE (zero)
- Consideration**   When you use a unary sign as part of the expression following a relational or numeric operator, you must enclose the expression in parentheses; for example, IF ALPHA=(-RESULT). See “[Arithmetic expressions](#)” on page 45.

---

## *blocka*

- Description**     *Optional.* Contains zero or more statements that you want to execute only if *expression* is TRUE.

***blockb***

**Description**     *Optional.* Contains zero or more statements that you want to execute if the expression is FALSE.

**General considerations**

- ◆ Each of the statements IF, ELSE, and END must appear on a line by itself. Only a comment (separated by a colon) can follow the IF expression.
- ◆ If the expression is TRUE (non-zero), MANTIS executes the statements in *blocka* and resumes at the statement following END.
- ◆ If the expression is FALSE (zero), and an ELSE clause *is not* present, MANTIS ignores *blocka* and resumes at the statement following END.
- ◆ If the expression is FALSE (zero), and an ELSE clause *is* present, MANTIS executes the statements in *blockb* and resumes at the statement following END.
- ◆ See also “FOR-END” on page 226, “UNTIL-END” on page 478, “WHEN-END” on page 508, and “WHILE-END” on page 510.

**Example**

The following example shows how the IF-ELSE-END statements can be used to test for a specific value:

```
000110 .WHILE MAP<>"CANCEL"
000110 ..GET REC(CUST_NUMBER)EQUAL
000120 ..IF REC="FOUND"
000130 ...MESSAGE="PF1" TO UPDATE      'PF3' TO CANCEL"
000140 ...CONVERSE MAP
000150 ...WHEN MAP="PF1"
000160 ....UPDATE REC
000170 ....MSG="UPDATE COMPLETE"
000180 ...WHEN MAP="PF3"
000190 ....MSG="MAINTENANCE CANCELLED AT USER'S REQUEST"
000200 ...END
000210 ..ELSE
000220 ...MSG="CUSTOMER NOT FOUND"
000230 ..END
000240 ..CLEAR MAP
000250 ..MESSAGE=MSG
000260 ..CONVERSE MAP
000270 .END

.
.
.
```

---

# INSERT

The INSERT statement inserts a new record into an external file, a MANTIS file, a personal computer file, an RDM logical view, or a TOTAL DBMS view.

## INSERT (External file)

---

**INSERT** *file-name*[LEVEL=*n*]

---

---

### *file-name*

<b>Description</b>	<i>Required.</i> Specifies the name (as defined in a previously executed ACCESS statement) of an existing external file where you want to insert a record.
<b>Format</b>	A MANTIS symbolic name (see “ <a href="#">Symbolic names</a> ” on page 24)

---

### **LEVEL=*n***

<b>Description</b>	<i>Optional.</i> Specifies the buffer number that contains the record you want to insert.
<b>Default</b>	1
<b>Format</b>	Arithmetic expression that evaluates to a value in the range 1 through <i>m</i> , where <i>m</i> is the maximum buffer number, as defined in the corresponding ACCESS statement

### **Considerations**

- ◆ Only specify LEVEL=*n* when the *file-name* has buffers defined.
- ◆ MANTIS uses only the integer portion of *n*.

General considerations

- ◆ An external file open (when required) is issued on the first DELETE, GET, INSERT, or UPDATE.
- ◆ For INDEXED files, the contents of key data elements identify the record to be inserted.
- ◆ For SEQUENTIAL files, MANTIS inserts the record at the end of the file and returns the Relative Byte Address (RBA) of the inserted record in the associated reference variable.
- ◆ For NUMBERED files, the Relative Record Number (RRN) contained in the corresponding reference variable identifies the record to be inserted.
- ◆ MANTIS returns a text string in the variable called *access-name* that reflects the status of the operation:

Returned text string	Description
""	The insert was successful.
"LOCK" *	The password specified in the ACCESS statement for this file view is not valid for deletions or insertions.
"ERROR" *	MANTIS received an error status. Use the FSI function (see "FSI" on page 232) for more information. Possible causes: <ul style="list-style-type: none"><li>◆ A physical error occurred during record insertion.</li><li>◆ The file was full.</li><li>◆ For INDEXED and NUMBERED files, the record to be inserted already exists.</li><li>◆ The External file exit canceled the operation.</li></ul>
"NOTOPEN" *	The external file is not open.

\* Returned only when TRAP is in effect for the file.

- ◆ For extended external file status messages and Function Status Indicators (FSIs), see “[Extended status messages for MANTIS and external files](#)” on page 521.
- ◆ This statement can be affected by the External File Exit. See your Master User for details.
- ◆ See also “[DELETE](#)” on page 183, “[FSI](#)” on page 232, “[GET](#)” on page 234, and “[UPDATE](#)” on page 479.



Fields that are not defined on the File View Design facility are added with spaces. This may produce incorrect results when the missing fields are defined as numeric in other programs. Simply defining the field as the proper datatype and not referencing it will set the field to zero (see “[Automatic mapping](#)” on page 33).

### Example

In the following example, showing how an EXTERNAL FILE INSERT is coded, the first INSERT statement opens the file:

```
00020 .ACCESS RECORD( "INDEX", "SERENDIPITY", 16)
00030 .SCREEN MAP( "INDEX" )
00040 .CONVERSE MAP
00050 .COUNTER=1
00060 .WHILE MAP<>"CANCEL" AND COUNTER<17
00070 ..WHEN INDICATOR(COUNTER)="G"
00080 ...GET RECORD LEVEL=COUNTER
00090 ..WHEN INDICATOR(COUNTER)="D"
00100 ...DELETE RECORD LEVEL=COUNTER
00110 ..WHEN INDICATOR(COUNTER)="I"
00120 ...INSERT RECORD LEVEL=COUNTER
00130 ..END
```

# INSERT (MANTIS file)

INSERT *file-name*[LEVEL=*n*]

## *file-name*

<b>Description</b>	<i>Required.</i> Specifies the name (as defined in a previously executed FILE statement) of an existing file where you want to insert a record.
<b>Format</b>	A MANTIS symbolic name (see “ <a href="#">Symbolic names</a> ” on page 24)

## LEVEL=*n*

<b>Description</b>	<i>Optional.</i> Specifies the buffer number that contains the record you want to insert.
<b>Default</b>	1
<b>Format</b>	Arithmetic expression that evaluates to a value in the range 1 through <i>m</i> , where <i>m</i> is the maximum buffer number, as defined in the corresponding FILE statement

## Considerations

- ◆ Only specify LEVEL=*n* when the *file-name* has buffers defined.
- ◆ MANTIS uses only the integer portion of *n*.



### General considerations

- ◆ You must execute a corresponding FILE statement before the INSERT statement, using the valid INSERT/DELETE password.
- ◆ The Setpray Exit can affect this statement. See your Master User for details.
- ◆ MANTIS returns a text string in the variable called *file-name* that reflects the status of the operation:

Returned text string	Description
"	The insert was successful.
"LOCK" *	The password specified in the FILE statement for this file view is not valid for deletions or insertions.
"ERROR" *	<p>MANTIS received an error status. Use the FSI function (see "FSI" on page 232) for more information. Possible causes:</p> <ul style="list-style-type: none"> <li>◆ A physical error occurred during record insertion.</li> <li>◆ The file was full.</li> <li>◆ The record to be inserted already exists.</li> <li>◆ SETPRAY Exit canceled the operation.</li> </ul>
"DATA" *	A data conversion error occurred while inserting the record.

\* Returned only when TRAP is in effect for the file.

- ◆ For extended MANTIS file status messages and Function Status Indicators (FSIs), see "Extended status messages for MANTIS and external files" on page 521.
- ◆ See also "DELETE" on page 183, "FSI" on page 232, "GET" on page 234, and "UPDATE" on page 479.

### Example

The following example shows how a record is inserted in an internal file after the corresponding FILE statement:

```
00010 ENTRY INDEX
00020 .FILE RECORD( "INDEX", "SERENDIPITY" )
00030 .SCREEN MAP( "INDEX" )
00040 .GET RECORD
00050 .WHILE RECORD<>"END" AND MAP<>"CANCEL"
00060 ..CONVERSE MAP
00070 ..WHEN MAP="PF1"
00080 ...INSERT RECORD
00090 ..WHEN MAP="PF2"
00120 ...UPDATE RECORD
00130 ..END
00140 ..GET RECORD
00150 .END
00170 EXIT
```

## INSERT (Personal computer file)

---

**INSERT** *file-name*[LEVEL=*n*]

---

### *file-name*

<b>Description</b>	<i>Required.</i> Specifies the name (as defined in a previously executed ACCESS statement) of an existing file that you want to insert.
<b>Format</b>	A MANTIS symbolic name (see “ <a href="#">Symbolic names</a> ” on page 24)

### LEVEL=*n*

<b>Description</b>	<i>Optional.</i> Specifies the buffer number that contains the record you want to insert.
<b>Default</b>	1
<b>Format</b>	Arithmetic expression that evaluates to a value in the range 1 through <i>m</i> , where <i>m</i> is the maximum buffer number, as defined in the corresponding ACCESS statement

### Consideration

- ◆ Only specify LEVEL=*n* when the *file-name* has buffers defined.
- ◆ MANTIS uses only the integer portion of *n*.

### General considerations

- ◆ For SEQUENTIAL files, MANTIS inserts the record at the end of the file. Because the Relative Byte Address is unavailable for inserts, the referenced variable is set to 0. MANTIS returns the RBA in the reference variable if the insert was successful.
- ◆ For NUMBERED files, the Relative Record Number (RRN) contained in the corresponding reference variable identifies the record to be inserted.
- ◆ A good way to initialize PC NUMBERED files is to insert a record with the highest RRN to be used. PC CONTACT initializes all records with lower Relative Record Numbers to binary zeros.

- ◆ MANTIS returns a text string in the variable called *access-name* that reflects the status of the operation:

Returned text string	Description
""	The insert was successful.
"LOCK" *	The password specified in the ACCESS statement for this file view is not valid for deletions or insertions.
"ERROR" *	MANTIS received an error status. Use the FSI function (see "FSI" on page 232) for more information. Possible causes: <ul style="list-style-type: none"><li>◆ A physical error occurred during record insertion.</li><li>◆ The disk is full.</li></ul>
"DATA" *	A data conversion error occurred while inserting the record.

\* Returned only when TRAP is in effect for the file.

- ◆ For extended personal computer file status messages and Function Status Indicators (FSIs), see "Extended status messages for MANTIS and external files" on page 521.
- ◆ With TRAP *name* OFF, an INSERT error results in a fault message. The fault message text contains the PC file name and the file status. If a data conversion error occurs, the fault message also contains the refer value for numbered files and the name of the field for which the data error occurred.
- ◆ With TRAP *name* ON, a fault does not occur. The file symbolic name contains the file status and the FSI built-in function can be used to obtain the fault message information.
- ◆ If you insert a new record where a record already exists, the new record replaces the existing record and the existing record is lost.
- ◆ Files residing on a personal computer can only be accessed by that personal computer user.
- ◆ If the data file where you want to insert records has not been previously defined, PC CONTACT creates the file when you perform your first insertion.

- ◆ For PC files, MANTIS uses buffering for GET NEXTs and INSERTs to increase the speed of uploads and downloads.

A common download program reads records from a mainframe file and issues many INSERTs against a PC file. These INSERTs are not sent to the PC until the buffer fills or some other event forces the sending of the buffer.

- ◆ The following events cause MANTIS to send the insert buffer to the PC. Avoiding them can improve performance and take full advantage of INSERT buffering.
  - Task termination.
  - Program termination.
  - EXIT from an externally done program containing the ACCESS statement for the PC (a special case of program termination).
  - COMMIT.
  - Screen I/O.
  - Perform of an external interface.
  - Change of I/O operation, such as changing from an INSERT to a GET, UPDATE, or DELETE.
  - Full buffer.
- ◆ Because of buffering, an INSERT error may not be detected until some time after the MANTIS program issues it, such as on the next CONVERSE. INSERT errors are trappable using the TRAP statement only after COMMITs and INSERTs. Testing the file status after a COMMIT determines if any of the inserts in the buffer sent to the PC as a result of the COMMIT failed. Testing the file status after an INSERT determines if any of the INSERTs in the buffer sent to the PC as a result of a full buffer condition failed. If the buffer has not yet been sent to the PC, the file status after an INSERT always indicates that it was successful. Only the INSERT that fills the buffer returns an error status.

- ◆ When INSERT buffering is in progress for more than one PC file, there is one INSERT buffer for each ACCESS. With the exception of a change of I/O operation and a full buffer, any event that causes an INSERT buffer to be sent to the PC, sends all INSERT buffers to the PC. The order in which the INSERT buffers are sent to the PC is not predictable. If an error occurs for an INSERT in one of the buffers, only examination of the PC files can determine which of the INSERTs from the other buffers actually succeeded.
- ◆ Because mainframe resources are recoverable using ROLLBACK/COMMIT, and PC resources are not, error situations can cause unsynchronized data to occur. Design your applications to rebuild PC files to preserve data integrity.
- ◆ When subsequent ACCESSes to a given PC file are encountered, all buffering is disabled for all ACCESSes to that PC at all DO levels to preserve data integrity.
- ◆ See also “DELETE” on page 183, “FSI” on page 232, “GET” on page 234, and “UPDATE” on page 479.

**Example**

The following example shows how a personal computer file is inserted:

```
00010 ENTRY PC_EXAMPLE
00020 .VIEW V_REC( "SAMPLE_VIEW" )
00030 .ACCESS PC_REC( "SAMPLE_PC_ONE" , "REMOVE" )
00040 .GET V_REC
00050 .WHILE V_REC<>"END"
00060 ..INSERT PC_REC
00070 ..GET V_REC
00080 .END
00090 .SHOW "FILE TRANSFER COMPLETE":WAIT
00100 EXIT
```

## INSERT (RDM logical view)

---

```
INSERT view - name [
  NEXT
  PRIOR
  FIRST
  LAST
] [LEVEL = n]
```

---



---

### view-name

<b>Description</b>	<i>Required.</i> Specifies the name (as defined in a previously executed VIEW statement) of the logical view where you want to insert a record.
<b>Format</b>	A MANTIS symbolic name (see “ <a href="#">Symbolic names</a> ” on page 24)

---

### NEXT/PRIOR/FIRST/LAST

<b>Description</b>	<i>Optional.</i> Specifies the location where the logical record is to be inserted relative to the current positioning.
<b>Format</b>	Must be coded exactly as shown
<b>Default</b>	NEXT
<b>Options</b>	<p><b>NEXT</b> Places the record in the logical view following the current record, provided the keys are the same. If the keys are different or if the current position has not been established, INSERT...NEXT operates as INSERT...LAST.</p> <p><b>PRIOR</b> Places the record in the logical view before the current record, provided the keys are the same. If the keys are different or if no current record has been established, INSERT...PRIOR operates as INSERT...FIRST.</p> <p><b>FIRST</b> Places the record in the logical view so that subsequent GET...FIRST commands using the same key values retrieves this record.</p> <p><b>LAST</b> Places the record in the view so that a subsequent GET...LAST command using the same key values retrieves this record.</p>

LEVEL=*n*

Description	Optional. Specifies the buffer number that contains the record you want to insert.
Default	1
Format	Arithmetic expression that evaluates to a value in the range 1 through <i>m</i> , where <i>m</i> is the maximum buffer number, as defined in the corresponding VIEW statement

Considerations

- ◆ Only specify LEVEL=*n* when the *view-name* has buffers defined.
- ◆ MANTIS uses only the integer portion of *n*.

General considerations

- ◆ If you don't read a record before inserting, then RDM inserts relative to the current record position.
- ◆ You must execute a corresponding VIEW statement before you can execute the INSERT statement.
- ◆ MANTIS returns a text string in the variable called *view-name* that reflects the status of the operation:

Returned text string	Description
" "	The insert was successful.
"DUPLICATE" *	A record with the same key already exists.
"ERROR" *	MANTIS received an error status. Use the FSI function (see "FSI" on page 232) for more information. Possible causes: <ul style="list-style-type: none"><li>◆ A physical error occurred during database access.</li><li>◆ You tried to perform an invalid function on the user view.</li></ul>
"LOCK" *	You do not have permission to insert logical records in the logical view.

\* Returned only when TRAP is in effect for the file.



- ◆ RDM logical view INSERT sets three status functions to the application program that indicate processing results—FSI, ASI, and VSI. FSI indicates the success or failure of your command. ASI indicates the status of each field in the logical record. VSI indicates the highest field status within the logical record. For a complete discussion of these status functions, see “[Status functions](#)” on page 517.
- ◆ If the logical record to be inserted is uniquely keyed, and if the value of the keys to be inserted already exists in the database, the insert fails. If TRAP is ON, MANTIS returns “DUPLICATE”; otherwise, MANTIS displays an error message and halts execution.
- ◆ If your DBA specified ordering in the logical view definition or if the physical DBMS does not allow program control of ordering, MANTIS ignores the specification on the INSERT statement.
- ◆ If TRAP is OFF for this logical view and MANTIS receives a failure status from RDM logical view, MANTIS issues a RESET to ensure database integrity. If TRAP is ON and “ERROR” is returned, and the MANTIS program does not perform the RESET, then it is possible that MANTIS did only part of the insert.
- ◆ Your DBA can disallow insertions. If so, MANTIS returns the “LOCK” status if TRAP is in effect. If TRAP is not in effect, MANTIS displays a message and halts execution.
- ◆ If you have issued an INSERT since the last terminal I/O, MANTIS automatically issues a COMMIT prior to any further terminal I/O.
- ◆ See also “[ASI](#)” on page 93, “[DELETE](#)” on page 183, “[FSI](#)” on page 232, “[GET](#)” on page 234, “[UPDATE](#)” on page 479, and “[VSI](#)” on page 505.

**Example**

The following example shows how an RDM LOGICAL VIEW INSERT is coded:

```
00010 VIEW CUSTOMER( "CUST" )
00020 SCREEN MAP( "CUST_UPDATE" )
00030 SHOW"ENTER CUSTOMER NUMBER: "
00040 OBTAIN CUST_NO
00050 GET CUSTOMER(CUST_NO)
00060 IF CUSTOMER<>"FOUND"
00070 .INPUT_OK=FALSE
00080 .UNTIL INPUT_OK
00090 ..CONVERSE MAP
00100 ..DO EDIT_INPUT
00110 ..WHEN EDIT_OK
00120 ...INPUT_OK=TRUE
00130 ..END
00140 .END
00150 .INSERT CUSTOMER
00160 .SHOW"CUSTOMER INFORMATION INSERTED"
00170 ELSE
00180 .SHOW"CUSTOMER ALREADY EXISTS"
00190 END
```

## INSERT (TOTAL file view)

INSERT file - name	FIRST	[LEVEL = n]
	<u>LAST</u>	
	BEFORE <i>refer</i>	
	AFTER <i>refer</i>	

### file-name

<b>Description</b>	<i>Required.</i> Specifies the name (as defined in a previously executed TOTAL statement) of a TOTAL file view into which you want to insert a record.
<b>Format</b>	A MANTIS symbolic name (see “ <a href="#">Symbolic names</a> ” on page 24)

### LEVEL=n

<b>Description</b>	<i>Optional.</i> Specifies the buffer number that contains the record you want to insert.
<b>Default</b>	1
<b>Format</b>	Arithmetic expression that evaluates to a value in the range 1 through <i>m</i> , where <i>m</i> is the maximum buffer number, as defined in the corresponding TOTAL statement

### Considerations

- ◆ Only specify LEVEL=n when the *file-name* has buffers defined.
- ◆ MANTIS uses only the integer portion of *n*.

**FIRST/LAST/BEFORE *refer*/AFTER *refer***

---

**Description**     *Optional.* Specifies the location where the logical record is to be inserted relative to the current positioning.

**Default**            LAST

**Options**            FIRST   Inserts the record at the beginning of the set.

                          LAST   Inserts the record at the end of the set.

                          BEFORE *refer*   Inserts the record prior to the one specified by *refer*.

                          AFTER *refer*   Inserts the record after the one specified by *refer*.

**Consideration**   Applicable to variable-entry files only.

---

**refer**

- Description** *Optional.* Identifies the position in the file where MANTIS inserts the record.
- Format** 4-character text expression that must be coded exactly as shown
- Consideration** You can save and restore a *refer* using a text variable of at least four characters in length. The *refer* is always held in the Reference Variable name found on the TOTAL File View Design Facility menu. For more details, refer to *MANTIS Facilities, OS/390, VSE/ESA*, P39-5001.

**General considerations**

- ◆ You must execute a corresponding TOTAL statement before you can execute the INSERT statement.
- ◆ A TOTAL file open is issued on the first DELETE, GET, INSERT, or UPDATE.
- ◆ MANTIS returns a text string in the variable called *file-name* that reflects the status of the operation:

Returned text string	Description
"	The insert was successful.
"DUPLICATE" *	A record with the same key already exists.
"LOCK" *	The password specified in the TOTAL statement is invalid.
"NOTFOUND" *	The variable-entry chain set with the requested key does not exist.
"NOTOPEN" *	The TOTAL view is not open.
"NOTAVAL" *	The TOTAL file or view is not open.

\* Returned only when TRAP is in effect for the file.

- ◆ See also "**DELETE**" on page 183, "**GET**" on page 234, and "**UPDATE**" on page 479.

**Example**

The following example shows how a TOTAL INSERT is coded with the corresponding TOTAL statement preceding the INSERT statement:

```
00020 TOTAL BILL("BOM", "ASSEMBLY", 8)
```

```
00100 INSERT BILL BEFORE BILL_REFER(BUFFER-1)LEVEL=BUFFER
```

# INT

The INT function returns the integer value of a where a is any arithmetic expression.

**INT(a)**

**a**

**Description**     *Required.* Specifies any valid arithmetic expression.

**General consideration**

See also the **ROUNDED** parameter under “**LET (Numeric (BIG/SMALL) variables)**” on page 308.

**Examples**     The following examples show how the INT function returns the integer portion of arithmetic expressions.

Example	Results	Comments
INT ( 45 )	45	
INT ( 45 . 5 )	45	
INT ( -45 . 5 )	-45	
INT ( - . 5 )	0	

---

## INTERFACE

The INTERFACE statement specifies an interface that your program accesses.

---

### INTERFACE

*name1*(*[library1:]interface-name1,password1*[,PREFIX][,n1])

*[,name2*(*[library2:]interface-name2,password2*[,PREFIX][,n2])...]

---



---

### *name*

- |                      |   |
|----------------------|---|
| <b>Description</b>   | <i>Required.</i> Specifies the name used to refer to the interface in subsequent CALL statements. |
| <b>Format</b>        | A MANTIS symbolic name (see “ <a href="#">Symbolic names</a> ” on page 24)                        |
| <b>Consideration</b> | When the symbolic name is previously defined, MANTIS bypasses this definition.                    |

---

### *[library:]interface-name*

- |                    |   |
|--------------------|---|
| <b>Description</b> | <i>Required.</i> Specifies the name of the interface profile as you saved it during interface design. |
| <b>Format</b>      | 1–33 character text expression that evaluates to a valid interface name                               |

### Considerations

- ◆ If the interface is in another user’s library, you can access it by specifying the name of the user, followed by a colon and the interface name, as follows:  
  
*[library:]interface-name*
- ◆ If this parameter is used, the colon (:) is required.
- ◆ If the interface resides in your library, you can specify only the interface name.
- ◆ If you want this entity to be HPO bound, the library name is required even if it is your own library.
- ◆ This parameter is translated to uppercase upon execution of your program.

---

**password**

**Description**     *Required.* Specifies the password valid for the interface profile.

**Format**            1–16 character text expression

---

**PREFIX**

**Description**     *Optional.* Specifies whether MANTIS places the symbolic name and an underscore before all field names associated with this interface. See the FILE statement for a full explanation of prefixing.

**Format**            Must be coded exactly as shown

---

***n***

**Description**     *Optional.* Specifies how many buffers MANTIS should allocate to this interface.

**Default**            1

**Format**            Arithmetic expression that evaluates to a value in the range 1–254

**Considerations**

- ◆ MANTIS uses only the integer portion of *n*.
- ◆ When you use the *n* parameter to indicate buffering, add the LEVEL=*n* option to the CALL statement.



## General considerations

- ◆ Consult your Master User before using the INTERFACE statement.
- ◆ MANTIS translates the design names on the INTERFACE statement (*library* and *interface-name*) to uppercase upon execution of your program. MANTIS does not translate *password* to uppercase.
- ◆ See also “CALL” on page 137.

## Example

The following example shows an INTERFACE statement accessing an interface within a program:

```
00020 INTERFACE MASTER("CUSTOMERS", "ALIBABA", 10)
00030 SCREEN MAP("CUSTOMERS")
00040 WHILE MAP<>"CANCEL"
00050 .CONVERSE MAP
00060 .CALL MASTER("GET",keyfield) LEVEL=2
00070 END
```

---

# KANJI (Kanji users only)

The Kanji statement names and specifies dimensions for DBCS (Double Byte Character Set) variables and arrays.

---

```
KANJI name1[([n1,] length1)]  
[,name2[([n2,] length2 ) . . .]
```

---

---

## name

- Description**     *Required.* Specifies the name of the KANJI (DBCS) variable.
- Consideration**   When the symbolic name is previously defined, MANTIS bypasses this definition.

---

## n

- Description**     *Optional.* Specifies the number of elements in a DBCS array.
- Format**           Arithmetic expression that evaluates to a value in the range 1–255
- Considerations**
  - ◆ MANTIS rounds *n* to an integer value.
  - ◆ If not specified, *name1* is a KANJI scalar.

---

***length***

<b>Description</b>	<i>Optional.</i> Specifies the maximum length (in characters) of each KANJI element. The BYTE length will be twice <i>n</i> .
<b>Format</b>	Arithmetic expression that evaluates to a value in the range 1–127
<b>Default</b>	8

**Consideration** MANTIS rounds *length* to an integer value.

**General considerations**

- ◆ A KANJI variable contains a zero-length string (K"") upon initial definition.
- ◆ MANTIS accepts only as many characters in a KANJI variable as you specify in the KANJI statement.
- ◆ The following MANTIS statements allow KANJI literals and variables to be specified as parameters:

CALL	HEAD	POINT
DEQUEUE		SHOW
ENQUEUE	LET	SIZE
GET		
- ◆ The following MANTIS statements allow KANJI variables, but not KANJI literals, to be specified as parameters:

CHAIN	ENTRY-EXIT	DO	OBTAIN
-------	------------	----	--------
- ◆ See also “BIG” on page 134, “DBCS (Statement)(Kanji users only)” on page 181, “MIXD” on page 327, “MIXM” on page 328, “MIXMODE” on page 329, “MIXT” on page 331, “SMALL” on page 415, and “TEXT” on page 457.

**Example**

The following example shows how a Kanji statement names and specifies dimensions for DBCS variables:



---

In this example, < indicates SO or Shift-out, > indicates SI or Shift-in.

---

```
00010 KANJI FIELDK(5),ARRAYK(3,20)
00020 FIELDK=K"  %%  ":ARRAYK(1)=K"  %%% " :ARRAYK(2)=G  "<%%%">"
00030 SCREEN MAP("KANJI_MAP","PSW")
00040 WHILE MAP<>"CANCEL"
00050
00060
00070
      .
      .
      .
```

---

# KEY

The KEY function returns a text string that identifies the key you pressed in response to a CONVERSE, OBTAIN, PROMPT, or WAIT statement.

---

## KEY

---

### General considerations

- ◆ Possible KEY values are:
  - “PF1” through “PF24”
  - “PA1”
  - “PA2” (only if your Master User has assigned an alternate CANCEL key)
  - “CANCEL” (equal to PA2 or another installation-specified key)
  - “CLEAR”
  - “ENTER”
  - “PEN”
- ◆ Use KEY for identifying the key pressed on an unformatted screen.
- ◆ The following explain dependencies of KEY and CLEAR:
  - The CLEAR *mapname* statement sets the value of the *mapname* variable to an empty string (“”), but does not affect the KEY function.
  - The CLEAR statement, without a *mapname*, does not affect the value of any *mapname* variable, but sets KEY to “CLEAR”.
  - Pressing the CLEAR key on the terminal sets KEY to “CLEAR”.

- ◆ KEY represents the physical key pressed, and is not affected by entry into the key simulation field.



CANCEL is normally returned to the MANTIS program when the PA2 key is pressed, but your Master User can change this.

- ◆ See also “CLEAR” on page 145, “CONVERSE” on page 157, “OBTAIN” on page 341, “PROMPT” on page 376, and “WAIT” on page 506.

Example

The following example shows how the KEY function can be used to test for a user response on the keyboard:

```
00010 DO INITIALIZATION
00020 HEAD"DISPLAY EMPLOYEE"
00030 CLEAR
00040 SHOW"ENTER EMPLOYEE NUMBER : (ENTER/CANCEL)"
00050 OBTAIN EMPLOYEE_NUMBER
00060 WHILE KEY<>"CANCEL"
00070
00080
      .
      .
      .
```

Example	Results	Comments
KEY	"PF12"	
KEY(1,2)	"PF"	Substringing permitted.
KEY	"CANCEL"	May be installation-defined. See your Master User.
CLEAR mapname		
KEY	"ENTER"	CLEAR mapname does not affect KEY value.
CLEAR		
KEY	"CLEAR"	CLEAR without a mapname sets KEY value to "CLEAR".

---

# KILL

The KILL command terminates a program listing (line editor), a program currently paused, or a program in a loop. Whenever a program pauses by executing a WAIT, OBTAIN, PROMPT, or CONVERSE statement, or MANTIS has just issued the message “POTENTIAL PROGRAM LOOP ENCOUNTERED”, you can stop program execution by entering KILL.

---

## KILL

---

### General considerations

- ◆ Use the TAB key to place the cursor in the correct position on the bottom line (as shown in the examples below).
  - Unformatted screen: bottom-left (unsolicited input field) or bottom-right (key simulation field)
  - Formatted screen: (converse) key simulation field
- ◆ You can reexecute your program by issuing the RUN statement if you are in programming mode.
- ◆ Your Master User can change or disable the keyword KILL.
- ◆ KILL cannot be used if the screen has a Protect Bottom Line or Full Display attribute.
- ◆ See also “CONVERSE” on page 157 and “OBTAIN” on page 341.

## Examples

### Unformatted Screen (OBTAIN, PROMPT, SHOW, WAIT)



### Formatted Screen (CONVERSE)





# LANGUAGE (Function)

The LANGUAGE function returns the current language setting.

## LANGUAGE

### Example

When the current language code is “ENU” (US English). For a complete list of language codes, see the table in the following section.

Example	Results	Comments
LANGUAGE	"ENU"	
LANGUAGE ( 1 , 2 )	"EN"	Substringing permitted.

# LANGUAGE (Statement)

The LANGUAGE statement assigns a language code to the current task. The language code determines which messages, facility screens and help prompts are displayed on the terminal.

LANGUAGE=*t*

- t*

Description

Required. Specifies a text expression identifying the language code to be used.
- Format

An expression evaluating to 0 or 3 characters
- Options

The following table contains the MANTIS language codes:

Code	Language	Code	Language
AFR	Afrikaans	ELL	Greek
ARA	Arabic	ENA	Australian English
BEL	Byelorussian	ENG	UK English
BGR	Bulgarian	ENP	US English (uppercase)
CAT	Catalan	ENU	US English (upper/lower case)
CHT	Traditional Chinese	ESL	Spanish Latin American
CHS	Simplified Chinese	ESP	Spanish
CSY	Czech	FIN	Finnish
DAN	Danish	FRA	French
DEU	German	FRB	Belgian French
DES	Swiss German	FRC	Canadian French

Code	Language	Code	Language
FRS	Swiss French	RMS	Rhaeto-Romanic
GAE	Irish Gaelic	ROM	Romanian
HEB	Hebrew	RUS	Russian
HRV	Croatian	SKY	Slovakian
HUN	Hungarian	SLO	Slovenian
ISL	Icelandic	SQI	Albanian
ITA	Italian	SRB	Serbian (Cyrillic)
ITS	Swiss Italian	SRL	Serbian (Latin)
JPN	Japanese	SVE	Swedish
KOR	Korean	THA	Thai
MKD	Macedonian	TRK	Turkish
NLD	Dutch	UKR	Ukrainian
NLB	Belgian Dutch	URD	Urdu
NON	Norwegian Nynorsk	U01	User-defined 1
NOR	Norwegian Bokmal	U02	User-defined 2
PLK	Polish	U03	User-defined 3
PTB	Brazilian Portuguese	U04	User-defined 4
PTG	Portuguese		

## General consideration

When the value, *t*, is NULL (0 characters), MANTIS uses the language specification on the user profile. If the user profile does not contain a language code, the installation default is used (see your Master User for this value).

## Examples

- ◆ The following example sets the language code to JPN (Japanese):

```
LANGUAGE= "JPN"
```

- ◆ The following example sets the language code back to the user's profile language code, or installation default if the user does not have a language code defined:

```
LANGUAGE= " "
```

# LET (Numeric (BIG/SMALL) variables)

The LET statement assigns a value to a variable (or variables) or an array (or arrays). MANTIS evaluates the expression and sets the variable or array equal to that value.

$$[LET] \ v \begin{bmatrix} (i) \\ (i, j) \end{bmatrix} [ROUNDED(n)] = e1 [, e2, e3 . . . ]$$

## LET

**Description** *Optional.* Identifies the statement. When “LET” is omitted, the statement begins with the symbolic name, *v*.

### *v*

**Description** *Required.* Specifies a variable or a subscripted array element.

**Format** A MANTIS symbolic name (see “[Symbolic names](#)” on page 24)

**Consideration** If you have not provided an explicit definition of the variable, *v*, MANTIS assigns a BIG type.

### *i*

**Description** *Optional.* Indicates the occurrence of the named variable within the array. Required if *v* is a 1-dimensional array.

**Consideration** MANTIS rounds *i* to the nearest integer.

### *i,j*

**Description** *Optional.* Indicates the occurrence of the named variable within the matrix. Required if *v* is a 2-dimensional array.

**Consideration** MANTIS rounds *i,j* to the nearest integer.

---

**ROUNDED(*n*)**

**Description**     *Optional.* Specifies the number of decimal digits (*n*) to which you want the number carried out during the calculation (and before the number is assigned to a variable).

**Format**            Integer from 0–6, inclusive

**Considerations**

- ◆ Used when performing calculations whose results are real numbers.
- ◆ If ROUNDED is not used, two variables that seem to be equal (when displayed by SHOW statements) can actually compare unequal due to internal floating-point representation.
- ◆ Use the ROUNDED option when you want to control fractional results; for example, in currency calculations.
- ◆ See also “INT” on page 294.

## ***e1 [,e2,e3...]***

**Description**     *Required.* Specifies an expression to be evaluated. MANTIS assigns that value to the variable or array element on the left of the equal sign.

**Format**            Arithmetic expression

**Consideration** If you code more than one expression, an array element must also be coded on the left of the equal sign. MANTIS assigns values to sequential ascending elements of the array beginning with the element specified by *v* in row major order (see BIG and SMALL in “[Numeric data](#)” on page 41).

### **General consideration**

You may also use the CLEAR *v* statement to set all elements of the variable *v* to zero.

## Examples

In the following examples, assume newly-defined variables:

```
10 BIG A,B(3),C(4,2)
```

Example	Results	Comments
A=10	10	
A=.1E3	100	MANTIS E-notation.
A ROUND(2)=PI	3.14	
A = ("XXX"="YYY")	0	Value of expression is FALSE (0).
B(1)=10	10	Other elements remain 0.
B(1)=10,20,30	B(1)=10 B(2)=20 B(3)=30	
C(2,2)=3	C(2,2)=3	Other elements remain unchanged.
C(1,1)=0,1,2,3,4,5,6,7	C(1,1)=0 C(1,2)=1 C(2,1)=2 C(2,2)=3 C(3,1)=4 C(3,2)=5 C(4,1)=6 C(4,2)=7	Row major order.
A=1/2	0.5	
A=A+1	1	Initially, A=0.
CLEAR B	B(1)=0 B(2)=0 B(3)=0	

---

# LET (TEXT/KANJI/DBCS variables)

---

$$[LET]v \begin{bmatrix} (x) \\ (x,y) \\ (i,x,y) \end{bmatrix} = e1 [,e2,e3 . . .]$$

---

<b>v</b>	
<b>Description</b>	<i>Required.</i> Specifies a variable or a subscripted array element.
<b>Format</b>	A MANTIS symbolic name (see “ <b>Symbolic names</b> ” on page 24)

---

<b>x</b>	
<b>Description</b>	<i>Optional.</i> Indicates the starting position for the named variable on the left-hand side of the expression to receive data from the right-hand side of the expression.
<b>Format</b>	Numeric expression between 1 and the maximum size specified for the variable, v (inclusive), or between the opposite of the current size and -1, for negative subscripting
<b>Considerations</b>	
<ul style="list-style-type: none"><li>◆ Used with text and DBCS data.</li><li>◆ MANTIS rounds x to an integer value.</li><li>◆ If x is greater than the current length, MANTIS pads the variable, v, with blanks up to and including position x-1.</li></ul>	



---

***y***

- Description**     *Optional.* Indicates the last position for the named variable on the left-hand side of the expression to receive data from the right-hand side of the expression.
- Format**            A numeric expression between 1 and the maximum size specified for the variable , *v* (inclusive). Or, for negative subscripting, a numeric expression between the opposite of the current size of *v* and -1. *y* must be greater than or equal to *x*; for example, the (*x,y*) pairs (1,4) and (-4,-1) are valid.

**Considerations**

- ◆ Used with text and DBCS data.
- ◆ MANTIS rounds *y* to an integer value.
- ◆ The value of the second subscript (*y*) does not normally affect the resulting current size of the variable on the left-hand side, *v*, except where *y* is greater than the previous current length. In this case, MANTIS changes the current length to the value of *y*. Using two negative subscripts does not affect the resulting current size of the variable.

---

***i***

- Description**     *Optional.* Indicates the occurrence of the element within the TEXT array; for example, TEXT B(5,8), B(3) identifies the third element of the array. Used only for TEXT arrays, where it is required.
- Consideration**    MANTIS rounds *i* to an integer value.

**e1 [,e2,e3...]**

**Description**     *Required.* Specifies an expression or expressions to be evaluated. MANTIS assigns the value of the expression to the variable or array element on the left of the equal sign.

**Format**            Valid text or DBCS expression

**Consideration** If you code more than one expression, an array element must also be coded on the left of the equal sign. MANTIS assigns values to sequential ascending elements of the array, beginning with the element specified by *v*, in order.

For example:

```
TEXT X(10,2),X (1)= "AA" ,"BB" ,"CC" ,"DD" ,"EE" ,"FF"
```

is equivalent to:

```
X (1) = "AA"  
X (2) = "BB"  
X (3) = "CC"  
X (4) = "DD"  
X (5) = "EE"  
X (6) = "FF"
```

## General considerations

- ◆ You may use CLEAR to set all elements of the variable *v* to a zero-length string (NULL or ""). You may also use the PAD and UNPAD statements to add or remove multiple characters from either end of a string or substring.
- ◆ If you specify no subscripting (apart from the array occurrence), the current length of the variable on the left-hand side (LHS) of the expression is determined by the size of the string on the right-hand side (RHS) of the expression, up to the specified maximum of the variable (as shown in the example below). That is, the right-hand side is truncated to the maximum length of the receiving variable (in this case, 10):

```
TEXT A(10)
A=RHS
```

- ◆ In this case, the starting substring subscript (*x*) is supplied, but the ending substring subscript (*y*) is not supplied. See the following example:

```
A(X)=RHS
```

The expression value on the right-hand side (RHS) is inserted, starting from position *x* (if *x* is positive) or *currentlength+x* (if *x* is negative), until either the last character of RHS is inserted or the maximum size of the left-hand side (LHS) is reached.

If *x* is positive and is greater than the *currentlength+1*, blanks are inserted from position *currentlength+1* to *x-1*.

MANTIS may change the current length of LHS to represent the last character added from RHS.

- ◆ In this case,  $x$  and  $y$  (the starting and ending substring subscripts) are positive. See the following example:

$A(X, Y) = \text{RHS}$

The positions are relative from the beginning of the variable:

- $y-x+1$  characters are moved from RHS into position  $x$  of the LHS. For example,  $A(3,4)$  receives two characters;  $A(3,3)$  receives one character.
- If the RHS is shorter than  $y-x+1$  characters, then the LHS is padded with blanks through position  $y$ .
- If the RHS is longer or equal to  $y-x+1$  characters, then  $y-x+1$  characters are moved.
- If  $x >$  current length of the LHS, the LHS is padded with blanks through position  $x-1$ ; then MANTIS proceeds as above.

- ◆ In this case,  $x$  and  $y$  are negative. See the following example:

$A(X, Y) = \text{RHS}$

The positions are relative to the end of the variable's current length. For the following examples, let  $A$  have a current length of 10.

- $y-x+1$  characters are moved from the RHS into position  $\text{currentlength}+x+1$  of the LHS. For example,  $A(-4,-2)$  will move 3 (that is,  $3=(-2)-(-4)+1$ ) characters into  $A$ , starting 4 characters before the end (position 7).
- If the RHS is shorter than  $y-x+1$  characters, then the LHS is padded with blanks through position  $\text{currentlength}+y+1$ . For example, if the current length of  $A$  is a 10,  $A(-4,-2) = \text{"X"}$  will move "X" into position 7 and blanks into positions 8 and 9 ( $-3$  and  $-2$ ).
- If the RHS is longer than or equal to  $y-x+1$  characters, then  $y-x+1$  characters are moved. For example,  $A(-4,-2)$  will move 3 characters of the RHS, when the RHS is 3 or more characters long.

**Examples**

In the following examples:

- ◆ Assume newly-defined variables:

```
10 TEXT VAR1(10),VAR2(5,20),VAR3(80)
```

Example	Results	Comments
VAR1="MORNING"	"MORNING"	Size=7
VAR1="GOOD MORNING"	"GOOD MORNI "	Size=10 (RHS truncated)
VAR1="GOOD DAY"	"GOOD DAY"	Size=8
VAR1=NULL	" "	Size=0
VAR1="GOOD DAY"	"GOOD DAY "	Size=8
VAR1=VAR1(1,4)	"GOOD"	Size=4
VAR1(1,8)="GOOD MORNING"	"GOOD MOR "	Size=8; LHS limits characters.
VAR1="GOOD MORNING"	"GOOD MORNI "	Size=10
VAR1(6,9)="LUCK"	"GOOD LUCKI "	Size=10
VAR1(6,10)="LUCK "	" LUCK "	Size=10; blanks inserted from positions 1 through 5, and at position 10.
VAR1="GOOD MORNING"	"GOOD MORNI "	Size=10
VAR1(6,10)="LUCK "	"GOOD LUCK "	Size=10; pad blanks.
VAR1="GOOD MORNING"	"GOOD MORNI "	
VAR1(6,10)="LUCK "	"GOOD LUCK "	Size=10
VAR1(1,3)="BAD "	"BADD LUCK "	3 char insert
VAR1="GOOD MORNING"	"GOOD MORNI "	
VAR1(6,10)="LUCK "	"GOOD LUCK "	Size=10
VAR1(1,4)="BAD "	"BAD LUCK "	4 char insert
VAR1(1)="GOOD"	"GOOD"	Size=4

Example	Results	Comments
VAR1="AAAAAAAAA "	"AAAAAAAAA "	Size=10
VAR1(3)="BC "	"AABC "	Size=4
VAR1="AAAAAAAAA "	"AAAAAAAAA "	Size=10
VAR1(3,4)="BC "	"AABCAAAAAA "	Size=10
VAR1="GOOD "	"GOOD "	
VAR1(6,9)="LUCK "	"GOOD LUCK "	Pad blank
VAR1(7,7)="0 "	"GOOD LOCK "	
VAR2(2)="GREAT EXPECTATIONS "	"GREAT EXPECTATIONS "	Size=18
VAR2(2)=VAR(2)+"!! "	"GREAT EXPECTATIONS!! "	Size=20
VAR2(2)="GREAT EXPERIENCE "	"GREAT EXPERIENCE "	Size=16
VAR2(2,7)="EXPECTATIONS "	"GREAT EXPECTATIONS "	Size=18
VAR2(1,4)="GOOD MORNING "	" GOOD MORNING "	Size=15
		Pad 3 blanks
VAR2(1)="A ", "B ", "C ", "D ", "E "	VAR2(1) is "A "	Size=1 for each
	VAR2(2) is "B "	
	VAR2(3) is "C "	
	VAR2(4) is "D "	
	VAR2(5) is "E "	
VAR3="GOOD MORNING "	"GOOD MORNING "	
VAR3(-1)=" ' "	"GOOD MORNIN' "	Last character in current string.
VAR3="GOOD MORNING "	"GOOD MORNING "	
VAR3(-7,-5)="EVE "	"GOOD EVENING "	Seventh-to-last to fifth-to-last characters in current string.

Example	Results	Comments
VAR3="GOOD MORNING" VAR3(-7,-5)="E"	"GOOD MORNING" "GOOD E NING"	Seventh-to-last to fifth-to-last characters in current string. Short RHS padded with blanks.
VAR3="GOOD MORNING" VAR3(-7,-5)="EVE****"	"GOOD MORNING" "GOOD EVENING"	Seventh-to-last to fifth-to-last characters in current string. Long RHS truncated to three characters.
VAR3="GOOD MORNING" VAR3(-7)="EVENING" VAR3(-7)="AFTERNOON"	"GOOD MORNING" "GOOD EVENING" "GOOD AFTERNOON"	Last 7 characters in the string.

---

# LOG

The LOG function returns the natural logarithm of *a* where *a* is any valid arithmetic expression.

**LOG(*a*)**

*a*

**Description**     *Required.* Specifies any valid, positive arithmetic expression.

**General consideration**

See also “E” on page 210 and “EXP” on page 220.

**Example**     The following examples show how the LOG function is used to return the natural logarithm of an arithmetic expression:

Example	Results	Comments
LOG(1000)	6.90775528	
LOG(245)/LOG(10)	2.38916608	Equivalent to Log <sub>10</sub> 245
LOG(PI**2)	2.28945977	
LOG(0.5)	-.693147181	



---

# LOWERCASE

The LOWERCASE function converts a text string into lowercase.

---

## LOWERCASE(*t*)

---

*t*

**Description**     *Required.* Specifies any valid text expression that you want to convert to lowercase.

**Format**            A text expression

### General considerations

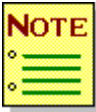
- ◆ The Customization TRCODE can affect this statement. See your Master User for details.
- ◆ The translation table used depends upon the current LANGUAGE setting.
- ◆ See also “[Text considerations](#)” on page 22, “[Text data](#)” on page 50, and “[UPPERCASE](#)” on page 493.

Examples

Example	Results	Comments
LOWERCASE ( "ABC" )	"abc"	
LOWERCASE ( "AbC % 123" )	"abc % 123"	
LOWERCASE ( "Æ" )	"æ"	Depends on your language setting and translation table. See your Master User.

- ◆ The following example shows how you can do a case insensitive compare on two text fields, A and B:

```
TEXT A,B
IF LOWERCASE(A)=LOWERCASE(B)
...(block of code for comparison equal)
END
```



Lowercase translation can be modified by your System Administrator for your native language.

# LUID

The LUID function returns a text string of 8 characters containing the VTAM logical unit ID (netname).

## LUID

### General Considerations

- ◆ The VTAM logical unit ID can be returned by the `TERMINAL` function when `TERMFUNC=VTAMID` is specified in the `C$OPCUST` macro. This function is available for those customers who had custom patches applied in previous releases of Mantis to return the VTAM logical unit ID (netname).
- ◆ The value returned by LUID varies depending on the operating environment:
  - For a physical terminal, this is the name by which this terminal is known to VTAM.
  - For ISC sessions, it is the name by which the session (or session group, if there are parallel sessions) is known to VTAM.
  - For MRO sessions, it is the name used by the connected region to log on to the interregion communication program. For a remote terminal, it is the name by which the terminal is known to the VTAM in the remote region. (For a remote terminal routed from a pre-CICS Transaction Server for OS/390 region, NETNAME is blank.)

### Example

Example	Results	Comments
LUID	"NMMAI032"	Sample LUID for CICS.
LUID	"BACK\$MAN"	Background task.
LUID	"DUMMY"	Batch MANTIS.
LUID(1,2)	"NM"	Substringing OK.

---

# MARK (SUPRA RDM users only)

The MARK statement obtains the current position of the logical view as established by the last GET, UPDATE, or INSERT statements. Before you can mark a view, you must open the view by processing the associated VIEW statement.

---

**MARK *view-name* AT *mark-name* [LEVEL=*n*]**

---

---

***view-name***

- |                    |  |
|--------------------|--|
| <b>Description</b> | <i>Required.</i> Specifies the name (as defined in a previously executed VIEW statement) of the RDM logical view you want to access. |
| <b>Format</b>      | A MANTIS symbolic name (see “ <a href="#">Symbolic names</a> ” on page 24)   |

---

***AT mark-name***

- |                      |  |
|----------------------|--|
| <b>Description</b>   | <i>Required.</i> Specifies a 4-character text variable where MANTIS saves the MARK information.          |
| <b>Format</b>        | A MANTIS symbolic name (see “ <a href="#">Symbolic names</a> ” on page 24)                               |
| <b>Consideration</b> | The AT clause in the GET statement relocates the logical view at the position set by the MARK statement. |

---

**LEVEL=*n***

<b>Description</b>	<i>Optional.</i> Specifies the buffer number of the record that is marked.
<b>Default</b>	1
<b>Format</b>	Arithmetic expression that evaluates to a value of 1 through <i>m</i> , where <i>m</i> is the maximum buffer number, as defined in the corresponding VIEW statement

**Considerations**

- ◆ Only specify LEVEL=*n* when the *view-name* has buffers defined.
- ◆ MANTIS uses only the integer portion of *n*.

**General considerations**

- ◆ If you issue the RELEASE statement, MANTIS releases all MARKs, and they are no longer valid.
- ◆ A MARK text variable contains internal information and must not be used in a SHOW command, or defined in a file, screen, or TOTAL view.
- ◆ You can control the number of outstanding MARKs in the following manner:
  - Reuse a prior MARK by executing the MARK statement with the prior value in the *mark-name* variable. MANTIS ignores the prior position and points previous variables to the new position.
  - Allocate a new MARK (while saving previous MARKs) by setting the *mark-name* variable to an empty string before executing the MARK statement.
- ◆ See also “GET” on page 234, “RELEASE (Function)” on page 378 “RELEASE (Statement)” on page 380, and “VIEW” on page 501.

### Example

The following example shows how the MARK statement obtains the current position of the logical view. Before you can mark a view, you must open the file by processing the associated VIEW statement.

```
00010 VIEW CUSTOMER("CUSTOMER_ACCOUNTS")
00015 TEXT CUST_MARK(4)
00020 GET CUSTOMER("R14148")
00030 MARK CUSTOMER AT CUST_MARK
00040 SHOW CUSTOMER_NUMBER
00050 GET CUSTOMER("X00070")
00060 SHOW CUSTOMER_NUMBER
00070 GET CUSTOMER AT CUST_MARK
00080 SHOW CUSTOMER_NUMBER
```

# MIXD

The MIXD function extracts DBCS (Double Byte Character Set) data from mixed-data.

## MIXD(*t*)

*t*

**Description**     *Required.* Specifies the text expression that can contain Shift-out and Shift-in codes and DBCS data.

### General considerations

- ◆ Using this function requires that the MIXMODE statement be set to ON. (See the MIXMODE statement.)
- ◆ If the text expression contains no DBCS data, an empty string is returned.
- ◆ See also “KANJI (Kanji users only)” on page 298, “MIXM” on page 328, “MIXMODE” on page 329, and “MIXT” on page 331.

**Example**     The following example shows how the MIXD function extracts DBCS data from mixed-data:



In this example, < means SO (Shift-out) and > means SI (Shift-in).

```
00010 MIXMODE ON
00020 TEXT ALPHA(20)
00030 KANJI GAMMA(20)
00040 ALPHA="A< 1 >BC< 2 >"
```

Example	Results	Comments
MIXD( ALPHA )	K"1 2"	Extracts only DBCS characters.

# MIXM

The MIXM function converts a DBCS expression to a mixed-data text string containing shift codes from DBCS data.

MIXM(*t*)

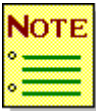
*t*

**Description**     *Required.* Specifies the DBCS expression to be converted.

**General considerations**

- ◆ Using this function requires that the MIXMODE statement be set to ON. (See the MIXMODE statement.)
- ◆ See also “KANJI (Kanji users only)” on page 298, “MIXD” on page 327, and “MIXT” on page 331.

**Example**     The following example shows how the MIXM function converts a DBCS expression to a mixed-data text string containing shift codes from DBCS data:



In this example < means SO (Shift-out) and > means SI (Shift-in).

```
00010 MIXMODE ON
00020 TEXT ALPHA(20)
00030 KANJI GAMMA(20)
00040 GAMMA=" 1 2 "
```

Example	Results	Comments
MIXM(GAMMA)	"< 1 2 >"	Results in TEXT expression containing mixed text with Shift-out/Shift-In.



## MIXMODE

The MIXMODE ON/OFF statement controls mixed-data. MIXMODE ON sets programs in mixed-data mode. MIXMODE OFF sets programs in non-mixed-data mode.

---

MIXMODE ON  
OFF

---

### ON

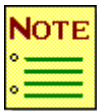
### OFF

**Description**     *Optional.* ON sets the program in Mix Data mode, and OFF sets the program in non-Mix Data mode.

**Default**            OFF

### General considerations

- ◆ MIXMODE ON means MANTIS processes both SO (Shift-out) and SI (Shift-in) in text variables. MIXMODE OFF means MANTIS does not check for shift codes when processing text expressions.



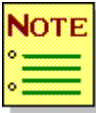

---

You should not use MIXMODE ON if you are using TEXT variables to hold arbitrary binary values.

---

- ◆ Optimum efficiency is obtained with MIXMODE OFF when the program does not handle mixed-data. Do not specify MIXMODE ON unless you are using text expressions containing shift codes.
- ◆ Mixed-data mode can also be set in Screen Design using the SO/SI attribute. Refer to *MANTIS Facilities*, *OS/390*, *VSE/ESA*, P39-5001.
- ◆ If MIXMODE is set to ON in your program, and that program executes an external DO, MIXMODE is set to OFF in the externally done program.
- ◆ See also “[KANJI \(Kanji users only\)](#)” on page 298, “[MIXD](#)” on page 327, “[MIXM](#)” on page 328, and “[MIXT](#)” on page 331.

**Example** The following example shows how the MIXMODE function is used to control the mixed-data mode:



In this example < means SO (Shift-out) and > means SI (Shift-in).

```
00010 MIXMODE ON
00020 TEXT A(80),B(80)
00030 A="abc< 1 2 >de< 3 >fg"
```

Example	Results	Comments
MIXMODE ON		
A-"< 2 >"	"abc< 1 >de< 3 >fg"	
MIXMODE OFF		
A-"< 2 >"	"abc< 1 2 >de< 3 >fg"	MANTIS is not doing special processing in looking for SO/SI strings.

# MIXT

The MIXT function extracts a SBCS (single byte character set) text string from text and mixed-data expressions.

## MIXT(*t*)

*t*

**Description**     *Required.* Specifies the text expression that can contain Shift-out and Shift-in codes, as well as DBCS data.

**Format**            A MANTIS symbolic name (see “Symbolic names” on page 24)

**General considerations**

- ◆ Using this function requires that the MIXMODE statement be set to ON. (See the MIXMODE statement.)
- ◆ See also “KANJI (Kanji users only)” on page 298, “MIXD” on page 327, “MIXM” on page 328, and “MIXT” on page 331.

**Example**            The following example shows how the MIXT function extracts a SBCS (single byte character set) text string from text and mixed-data expressions:



In this example < means SO (Shift-out) and > means SI (Shift-in).

```
00010 MIXMODE ON
00020 TEXT ALPHA(20),BETA(20)
00030 ALPHA="A< 1 >BC< 2 >"
```

Example	Results	Comments
MIXT( ALPHA )	"ABC"	Extracts only SBCS (Text) data

---

# MODIFIED

The MODIFIED function tests whether a specific field, any field within a map definition, or any field within the entire map set changed during the last physical I/O. Because zero evaluates to FALSE, you can use MODIFIED as a logical or arithmetic function.

---

**MODIFIED** { *(screen - name* [, *field - name*])  
(**TERMINAL**) }

---

---

## *screen-name*

- Description**    *Optional.* Specifies the name (as defined in a previously executed SCREEN statement) of the screen you are testing.
- Format**        A MANTIS symbolic name (see “Symbolic names” on page 24)
- Consideration** If you supply only *screen-name*, MANTIS returns the number of fields altered during the last physical I/O to the terminal. MANTIS returns 0 (FALSE) if no fields were modified.

---

## *field-name*

- Description**    *Optional.* Specifies the name of the field you are testing.
- Format**        A MANTIS symbolic name (see “Symbolic names” on page 24)
- Consideration** If you supply *field-name*, MANTIS returns TRUE (1) if, and only if, you alter the specified field. Otherwise, MANTIS returns FALSE (0).

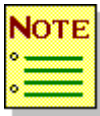
---

## TERMINAL

**Description**     *Optional.* Indicates if any field in the map set was modified.

### General considerations

- ◆ MANTIS resets modified field indicators only when you CONVERSE a screen for physical I/O. In other words, MANTIS does not reset field indicators when you converse a screen with the WAIT option. If you add the UPDATE option to a CONVERSE statement, MANTIS allows you to modify all unprotected fields in a map set.
- ◆ If the screen is defined in a caller and passed as a parameter on an external DO, both the screen and the field-name must be passed if modified (*map,field*) is used.
- ◆ See also the **KEEP MAP MODIFIED attribute** under “**General considerations for the ATTRIBUTE statement**” on page 110.



---

MANTIS evaluates 0 as FALSE and any other number as TRUE when the MODIFIED function is used in a conditional expression.

---

## Examples

- ◆ The following example determines the fields that were modified during the last CONVERSE statement:

```
00010 SCREEN CLIENT_INFO("CLIENT_INFORMATION")
00020 CONVERSE CLIENT_INFO
00030 IF MODIFIED(CLIENT_INFO)
00040 .DO CLIENT_PROCESSING:|ONE OR MORE FIELDS CHANGED
00050 END
```

- ◆ The following example uses the MODIFIED function as a logical or arithmetic function:

```
00010 ENTRY CLIENT_PROCESSING
00020 .FIELDS_TO_CHECK=MODIFIED(CLIENT_INFO)
00030 .I=1
00040 .LIMIT=SIZE(CLIENT_NAME,1)
00050 .WHILE I<=LIMIT AND FIELDS_TO_CHECK
00060 . .IF MODIFIED(CLIENT_INFO,CLIENT_NAME(I))
00070 ...|   perform any edit checks
00080 ...FIELDS_TO_CHECK=FIELDS_TO_CHECK-1
00090 ..END
00100 ..I=I+1
00110 .END
00120 EXIT
```

# NEXT

Use the NEXT statement to proceed immediately to the next conditional repeat in a FOR-END, UNTIL-END, or WHILE-END statement or to the next WHEN condition in a WHEN-END statement.

## NEXT

### General considerations

- ◆ In FOR-END, and WHILE-END statements, NEXT results in the repeat only if the respective condition is satisfied. If the condition is not satisfied, the statement after the END statement is executed.
- ◆ In UNTIL-END, NEXT results in the repeat only if the UNTIL condition is not satisfied. If the UNTIL condition is satisfied, the statement after the END statement is executed.
- ◆ In FOR-END statements, NEXT causes MANTIS to add the increment to the counter before comparing the counter and to perform the FOR-END termination checking (see “FOR-END” on page 226).
- ◆ In WHEN-END statements, NEXT causes MANTIS to drop through to the next WHEN condition or to the END statement (after the last WHEN in the program).
- ◆ NEXT is not required before the END statement. Executing the END statement is sufficient to repeat the FOR, WHILE, or UNTIL, based on condition checking.
- ◆ With nested logic statements, NEXT proceeds with the innermost enclosing FOR-END, UNTIL-END, WHEN-END, or WHILE-END statements where it occurs.
- ◆ See also “BREAK” on page 136 and “RETURN” on page 388.

### Examples

The following example shows how NEXT can be used with a FOR-END loop:

```

10 FOR L=1 TO MAXLINES
20 .IF NOT(MODIFIED(CUST_DETAIL)) <----If this condition is TRUE, then
30 . NEXT                               logic flow will continue to
40 .END                               statement 10.
50 .UPDATE CUSTOMER_FILE
60 END

```

---

# NOT

The NOT function returns TRUE (1) for an arithmetic expression if *a* evaluates to FALSE (0); otherwise, NOT returns FALSE (0).

---

**NOT(*a*)**

---

***a***

**Description**     *Required.* Specifies the value whose logical negation is to be returned.

**Format**             Specifies any valid arithmetic expression

**General considerations**

- ◆ Note that NOT is a function rather than an operator like AND and OR. See AND and OR operators in “Arithmetic expressions” on page 45.
- ◆ See also “ABS” on page 86, “FALSE” on page 221, “SGN” on page 399, and “TRUE” on page 472.



Examples

Example	Results	Comments
FLAG=TRUE		
NOT(FLAG)	0 (FALSE)	
A=3:B=3		
NOT(A=B)	0 (FALSE)	
A=3		NOT value for any non-zero is 0.
7+NOT(A)	7	
A=1:B=0		A and B are treated as Boolean values.
NOT(A OR B)	0 (FALSE)	

The following example shows how the NOT function can be used to test numeric values:

```
00010 IF NOT(A=3 OR J=1)
.
.
.
00100 END
```

The following example shows how the NOT function can be used to test a logic variable:

```
00010 BIG ERROR
00090 DO EDIT_CHECK(ERROR)
00100 IF NOT (ERROR)
.
.
.
00200 END
```

# NULL

The NULL function returns a null (zero-length) text or DBCS value (“”).

## NULL

### General considerations

- ◆ Do not confuse the null string result with null values or missing values associated with RDM user views or SQL variables.
- ◆ MANTIS does not provide RDM null support with this function.
- ◆ MANTIS does not provide support to SQL host variables with this function. Use indicator variables for null specification for SQL host variables.
- ◆ See also “FALSE” on page 221, “TRUE” on page 472, and “ZERO” on page 512.

### Example

The following example shows how the NULL function returns a zero-length string:

Example	Results	Comments
NULL	" "	

The following shows how you can use NULL as a text expression:

```
00010 TEXT A
00020 A=NULL
00030 WHILE A=NULL
.
.
.
00050 END
```

# NUMERIC

Use the NUMERIC function to determine if a text expression contains a valid number.

**NUMERIC** (*text-expression*)

## *text-expression*

**Description**     *Required.* Specifies the expression you want MANTIS to check.

**Format**            Text expression

### General considerations

- ◆ TRUE (1) is returned if the text expression contains a valid number; FALSE (0) is returned if it is not a valid number.
- ◆ A valid number is defined by the following rules:
  - May contain one decimal point (defined in your User Profile).
  - May contain commas (thousands separators) in valid positions, which are defined in your User Profile.
  - May have a leading or trailing sign character, but not both. The sign character may be separated from the rest of the number by blanks.
  - Must contain at least one numeral (0–9) without embedded spaces.
  - May not contain currency notation (for example, \$, £, or €) or E notation.

Examples

Example	Results	Comments
NUMERIC( "123,456.789" )	TRUE	
NUMERIC( "-.05" )	TRUE	
NUMERIC( "-432.876" )	TRUE	
NUMERIC( "- 432.876" )	TRUE	
ABC="-1234.55"	TRUE	
NUMERIC(ABC+"44" )		
NUMERIC ( " " )	FALSE	Empty string,
NUMERIC( "\$1234.55" )	FALSE	Contains "\$".
NUMERIC( "A33" )	FALSE	Contains alphabetic.
NUMERIC( "12-233-0323" )	FALSE	Embedded "-"

---

## OBTAIN

The OBTAIN statement gets data from an unformatted screen and assigns that input data to numeric and text variables. You can also use OBTAIN to retrieve unsolicited data (in the lower left corner) on a formatted screen if the OBTAIN follows a CONVERSE.

---

**OBTAIN v1, v2, v3,...**

---



---

**v**

**Description**     *Required.* Specifies the arithmetic, text, or DBCS variables or array elements you want to obtain.

**Format**            A MANTIS symbolic name, subscripted if an array (see “[Symbolic names](#)” on page 24)

### General considerations

- ◆ You can mix different types of variables in your list.
- ◆ When you enter values in response to an OBTAIN statement, separate the values with semicolons.
- ◆ If MANTIS already has data on the screen from a previous CONVERSE statement, and no intervening SHOW or PROMPT statement has been executed, the data you entered during the previous CONVERSE feeds the OBTAIN statement and no terminal interaction takes place.
- ◆ If the OBTAIN statement follows a SHOW, MANTIS waits until you key in data. MANTIS assigns these input data items to variables in the list until it exhausts the list or your input data. If MANTIS exhausts the input data before assigning values to all items in the list, the values of the remaining items on the list are unaltered.
- ◆ When lowercase input is accepted from the terminal (ATTRIBUTE (TERMINAL) = “LOWERCASE”), TEXT fields from the OBTAIN are left as they are entered. If you want translation on these fields, you must use the UPPERCASE function on them, for example:

```
OBTAIN NAME
NAME=UPPERCASE ( NAME )
```

- ◆ If you have DBCS language support, MANTIS accepts multiple DBCS fields from the command line. DBCS fields can be mixed with text and numeric fields.
- ◆ See also “CONVERSE” on page 157, “KEY” on page 301, “KILL” on page 303, “SHOW” on page 400, and “WAIT” on page 506.

## Examples

- ◆ The following example shows how OBTAIN gets data from a formatted screen:

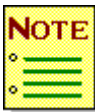
```
00005 FILE RECORD("DEMO_LIST", "PASS3")
00010 SCREEN MAP("DEMO")
00020 SHOW"ENTER KEY FOR INQUIRY"
00030 OBTAIN ACCT_NUMBER
00040 GET RECORD(ACCT_NUMBER)
00050 WHILE MAP<>"CANCEL"
00060 .CONVERSE MAP
00070 .OBTAIN ACCT_NUMBER
00080 .GET RECORD(ACCT_NUMBER)
00090 END
```

- ◆ The following example shows how an OBTAIN statement can be used with a SHOW statement for an unformatted screen:

```
00005 SHOW"PLEASE ENTER MONTH;DAY;YEAR"
00010 OBTAIN MONTH, DAY, YEAR
```

- ◆ During execution, the input would appear in the following format:

```
PLEASE ENTER MONTH;DAY;YEAR
4;2;91
```



---

You must use a semicolon (;) to separate data in response to an OBTAIN statement.

---

# ORD

Use the ORD function to return the numeric value of the first character EBCDIC code.

## ORD(*a*)

*a*

**Description**     *Required.* Specifies a text expression whose value you want returned.

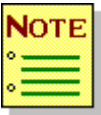
**Format**             Any text expression

### General considerations

- ◆ Only the first character of the text expression is returned as value.
- ◆ If *a* specifies a null string, ORD(*a*) returns -1.

### Example

Example	Results	Comments
ORD( "C" )	195	
ORD( "BECAUSE" )	194	Same as ORD("B").
ORD( " ." )	75	
ORD(NULL)	-1	Indicates null string.



The CHR and ORD functions depend on the machine architecture. Results will be different for code moved to an ASCII machine.

# OUTPUT

The OUTPUT statement routes output from the CONVERSE and SHOW statements and the LIST command.

OUTPUT {  
    SCREEN  
    PRINTER[VIA e]  
    SCREEN PRINTER [VIA e]

## SCREEN

- Description

Optional. Tells MANTIS that you want output from CONVERSE, PROMPT, SHOW, or LIST to be routed to your display terminal.
- Format

Must be coded exactly as shown

## PRINTER

- Description

Optional. Tells MANTIS that you want output from CONVERSE, PROMPT, SHOW or LIST to be routed to your designated printer.
- Format

Must be coded exactly as shown
- Consideration

The destination PRINTER is named by user profile, and the default printer ID is named by the PRINTER= statement, or by an installation exit.

## VIA e

- Description

Optional. Tells MANTIS the name of a printer exit program. VIA must be coded exactly as shown. The e must be a text expression that evaluates to a 1–8 character printer exit name. (If the first five characters are /MANX, the length is 5–9. Refer to *MANTIS Administration, OS/390, VSE/ESA*, P39-5005 for more information on printer exits.)

## Considerations

- ◆ If you don't specify a program, MANTIS either handles the output or passes it to the default printer exit program (if specified in your user profile).
- ◆ Contact your System Administrator or Master User for printer exits available at your site.



## General considerations

- ◆ The OUTPUT statement clears all output lines from the SHOW statement that WAIT or OBTAIN have not forced out to the terminal.
- ◆ When the OUTPUT PRINTER is in effect, but the OUTPUT SCREEN is not, the WAIT, OBTAIN, or CONVERSE statements simulate the condition of an `ENTER` key being pressed. MANTIS does not suspend the program. OUTPUT that is only to a printer does not reset the current SLOT and SLICE counters. Therefore, a program that runs while conversing to a screen can fail with POT or RPR errors when output is only sent to a printer.
- ◆ To have I/O return to the screen after you have sent data to the printer (via the OUTPUT PRINTER statement), add an OUTPUT SCREEN statement.
- ◆ Printer destination can be changed by the installation exit. See your Master User for details.
- ◆ See also “[PRINTER \(Statement\)](#)” on page 371.

## Examples

- ◆ The following example routes output to your display terminal. In programming mode, MANTIS automatically routes output to the display terminal unless you specify otherwise.  
`OUTPUT SCREEN`
- ◆ The following example routes output from SHOW and CONVERSE statements to the printer assigned to your terminal. You can print an entire program on an online printer using the OUTPUT PRINTER statement in immediate mode, followed by a LIST command. When MANTIS sends the program to the printer output queue, the output returns to your display terminal.  
`OUTPUT PRINTER`
- ◆ The following example routes output to your display terminal and printer:  
`OUTPUT SCREEN PRINTER`
- ◆ The following example routes output to your display terminal and to the printer, but also routes it to a printer exit program (named EXAMPLEA) for processing as that program directs:  
`OUTPUT SCREEN PRINTER VIA "EXAMPLEA"`

---

# PAD

The PAD statement allows you to pad either or both sides of a text or DBCS variable with a specified character.

---

```
PAD v [exp] [ AFTER
              ALL
              BEFORE ]
```

---

v

**Description**     *Required.* Represents the variable to be padded.

**Format**            A subscripted or unsubscripted MANTIS symbolic name

**Considerations**

- ◆ Must be a TEXT, KANJI, or DBCS variable.
- ◆ If the referenced variable is subscripted apart from the array occurrence (see the LET statement), the BEFORE, AFTER, and ALL options cannot be used. If you try to use the substring subscripts with one of these options, you receive an error message.
- ◆ If the referenced variable is not substring subscripted, MANTIS supplies the necessary pad characters to fill the variable to its maximum length.
- ◆ If the referenced variable has two substring subscripts, each subscript represents the boundaries of the pad operation. The leftmost boundary is marked by the first substring subscript; the rightmost boundary by the second substring subscript.
- ◆ If the referenced variable has one substring subscript, MANTIS assumes that the second (missing) substring subscript is equal to the maximum size of the variable. Therefore, the first substring subscript marks the starting point of the pad operation. With no second subscript, the end of the pad operation is the rightmost byte of the originally defined area for the variable.

---

**exp**

**Description**     *Optional.* Indicates a text (or DBCS) expression that represents the pad character.

**Format**            A 1–*n* character TEXT, KANJI, or DBCS expression

**Considerations**

- ◆ If you do not supply a value, MANTIS automatically uses spaces for the pad character.
- ◆ Only the first character of *exp* is used.

---

**AFTER**

**Description**     *Optional.* Specifies that padding occur on the right-hand side of the variable (trailing pad character). This is the default value.

---

**ALL**

**Description**     *Optional.* Indicates that padding occur on both sides of the variable (leading and trailing pad characters), centering the original text as much as possible.

**Consideration** If you specify the ALL option, the original value is centered within the variable, surrounded by the pad characters.

---

**BEFORE**

**Description**     *Optional.* Specifies that padding occur on the left-hand side of the variable (leading pad characters).

**General considerations**

- ◆ MANTIS automatically pads text and DBCS variables on the right with blanks before sending to an external file, TOTAL file, interface, or view.
- ◆ See also “**LET (TEXT/KANJI/DBCS variables)**” (left-hand side subscripting) starting on page 312, “**POINT**” on page 368, and “**UNPAD**” on page 474.

Examples

The following examples of code and results show how the PAD statement can be used with various symbols on either side of an expression. In the following examples:

```
00010 TEXT A(20),B(20)
00020 A="ABC"
```

Example	Results	Comments
PAD A "*" ALL	"*****ABC*****"	Centered, filled with "*".
PAD A BEFORE	"ABC"	Right-justified, filled with blanks.
PAD A	"ABC"	Left-justified, filled with blanks.
PAD B "?"	"????????????????"	Empty string filled with the "?" character.
PAD A "*" AFTER	"ABC*****"	
PAD A "-" BEFORE	"-----ABC"	
PAD A(3) "/"	"AB/////////"	Filled from position 3 to the end of the string with "/".
PAD A(7,11) "&"	"ABC&&&&&"	Filled from positions 4–6 with blanks, then filled from positions 7–11 with "&". (The current length is set to 11.)
PAD A "+"	"ABC++++++"	
PAD A(7,11) "&"	"ABC+++&&&&++++"	Middle of string filled with "&".

# PASSWORD

The PASSWORD function returns a text string containing the current password for the signed on user.

## PASSWORD

### General considerations

- ◆ PASSWORD always returns a text string 1–16 characters long.
- ◆ Statements containing the PASSWORD function do not HPO Bind. Use a variable or literal password to ensure bindability.
- ◆ See also “USER” on page 497.

### Example

The following example shows how the PASSWORD function is used to return a text string containing the current user password:

Example	Results	Comments
PASSWORD	"5P4S67"	
PASSWORD(1,4)	"5P4S"	Substringing permitted.

```
00010 ENTRY CUST_ENTRY
00020 .SCREEN MAP("CUST_ENTRY")
00030 .FILE REC("CUST_FILE",PASSWORD)
00040 .FILE RECX("CUST_FILE",PASSWORD,PREFIX)
00050 .CONVERSE MAP
00060 .WHILE MAP<>"CANCEL"
```

# PERFORM

The PERFORM statement invokes a user-written COBOL, Assembler, or PL/I target program without passing parameters to it. When you perform another program, your program can either return to MANTIS or transfer control to a new program. If you are a CICS user, you can also use the PERFORM statement to run a MANTIS or external transaction as a background task. You can also end MANTIS and delete the context. Control will not resume at the statement following the PERFORM the next time MANTIS is invoked. Control remains with the performed module and does not return to MANTIS.

## PERFORM *t*

<i>t</i>	
Description	<i>Required.</i> Specifies the text expression that represents the object of the PERFORM.
Options	Where the <i>t</i> is a text expression evaluating to one of the following: <ul style="list-style-type: none"><li>◆ “<i>program</i>”</li><li>◆ “<i>program/XCTL</i>”</li><li>◆ “<i>program/EXEC</i>”</li><li>◆ “[<i>trans-id</i>]/BACK,<i>user-id,password,program</i>[;<i>text-string</i>]”</li><li>◆ “<i>trans-id/EXTN</i>[,<i>text-string</i>]”</li><li>◆ “[<i>transaction code</i>]<i>b</i> [<i>text-string</i>]”</li><li>◆ “<i>transaction code/XCTL</i>[,<i>text-string</i>]”</li></ul>

## General consideration

The differences between a PERFORM and a CALL routine are:

CALL:

- ◆ Passes data in the predefined interface area.
- ◆ Runs in foreground only.
- ◆ Can only be a non-MANTIS program.
- ◆ Cannot do terminal I/O unless the MANTIS user is conversational CICS.
- ◆ Returns to the next MANTIS statement.

PERFORM:

- ◆ Passes data in the text expression.
- ◆ Runs in foreground or background.
- ◆ Can be a MANTIS or non-MANTIS program.
- ◆ Can do terminal I/O regardless of the MANTIS user.
- ◆ Can run outside of MANTIS and then return to the next MANTIS statement.

The following table summarizes the different variations of PERFORM, which are each described separately in the following pages:

Option	Context saved	How invoked	Target a MANTIS program?	COMMAREA passed? (CICS only) non-included programs	Does control return to next MANTIS statement
<i>program</i>	Yes, unless no <i>TRANSID</i> on EXEC CICS RETURN	CICS LINK, CALL for batch	No	Yes	Yes
/XCTL	No	CICS XCTL	No	Yes	No
/EXEC	Yes	CICS XCTL	No	Yes	Yes
/BACK	No	CICS START	Yes	No	Yes
/EXTN	No	CICS START	No	No	Yes



## PERFORM transfers control to another program without passing program variables

### PERFORM "*program*"

#### *program*

**Description**     *Required.* Specifies the name of the target program you want to invoke.

**Format**             Must be a text expression producing a text value of 1–8 characters in length and must not contain embedded blanks

#### Considerations

- ◆ Under CICS, the program must be defined in the PPT (Processing Program Table) or CSD (CICS System Definition File) under RDO (Resource Definition Online) and exist in the DFHRPL Load/Core Image Library.
- ◆ Under CICS, the target program is invoked via the “EXEC CICS LINK” command. MANTIS saves the TWA before control is passed to the target program. Therefore, the target program can use the TWA area as a workspace. MANTIS is still in storage. After subtask completion, CICS gives control back to MANTIS.
- ◆ Under CICS, if your target program runs in conversational mode, it returns to the next MANTIS statement after the PERFORM when you issue the “EXEC CICS RETURN” command.
- ◆ Under CICS, if your target program wants to do terminal I/O in pseudoconversational mode, it must do the following:  

```
EXEC CICS RETURN TRANSID (your-transid)
```

MANTIS then saves its context and return to CICS. CICS sends the next terminal input to *your-transid*.
- ◆ Under CICS, if MANTIS has been passed a COMMAREA by a previous task, it passes it to the target program.
- ◆ In batch, the target program is invoked via BALR 14,15. The program must exist in the STEPLIB/JOBLIB load library or sublibrary chain.
- ◆ MANTIS issues DBMS SINON and SINOF calls. The target program must not issue them. If it does so, FUNC or NACT DBMS statuses occur. See TOTAL and VIEW ON/OFF if your interface programs do sign ons or sign offs, or expect a certain state (on or off) when they are invoked.

## General considerations

- ◆ The object of the PERFORM can be any text expression. The literal shown here as an example is “*program*”.
- ◆ When you want to return to MANTIS at the statement following the PERFORM, enter code such as:

```
EXEC CICS START TRANSID (resX) TERMINAL (current terminal)
```

or

```
EXEC CICS RETURN TRANSID (resX)
```

where *resX* is a valid MANTIS resume transaction ID.

The transaction ID in MANTIS can be either paired or non-paired.

With paired IDs, there is an initial ID and a resume ID. If you start MANTIS with the initial ID, MANTIS always begins by executing your sign-on and facility programs. With the resume ID, MANTIS attempts to resume at the statement following the PERFORM. If you have paired IDs, and the ID you supply is not contained in the list of transaction ID pairs, MANTIS attempts to resume at the statement following the PERFORM.

With non-paired IDs, MANTIS attempts to resume at the statement following the PERFORM unless the task is a background task, in which case MANTIS begins by executing your sign-on and facility programs.

See your Master User for details on transaction IDs.

- ◆ In CICS, if you XCTL back to MANTIS, it reinitializes. It does NOT resume at the statement following the PERFORM.
- ◆ The PERFORM statement can be used to construct menu-driven systems where the menu and some system components are written in MANTIS. Performance-sensitive or existing components can remain as COBOL, Assembler, or PL/I, and can be invoked from MANTIS via the PERFORM statement.
- ◆ Batch MANTIS supports the CALL and PERFORM verbs using normal IBM linkage conventions.
- ◆ See also “CALL” on page 137, “DO” on page 206, and “RUN” on page 391.

**Example**

The following example shows how to use the PERFORM statement to invoke another program into your MANTIS program without trading variables:

```
00010 ENTRY FACILITY
00020 .SCREEN MAP( "MENU" )
00030 .CONVERSE MAP
00040 .WHILE MAP="CANCEL"
00050 ..WHEN OPTION=1 OR MAP="PF1"
00060 ...CHAIN"CUSTOMER_NAMES"
00070 ..WHEN OPTION=2 OR MAP="PF2"
00080 ...PERFORM"INVOICES"
00090 ..WHEN OPTION=3 OR MAP="PF3"
00100 ...PERFORM"/BACK,USR1,PSW1,ACCOUNTING:REPORTS"
00110 ..END
00120 .END
00130 END
```

**PERFORM transfers control to an external program without a return**

**PERFORM "program/XCTL"**

***program***

**Description**     *Required.* Specifies the name of the target program you want to invoke.

**Format**             Must be a text expression from 1–8 characters in length, and must not contain embedded blanks

**Considerations (CICS)**

- ◆ The program must be defined in the PPT (Processing Program Table) or CSD (CICS System Definition File) under RDO (Resource Definition Online) and exist in the DFHRPL Load/Core Image Library.
- ◆ The target program runs under the MANTIS transaction code until “CICS RETURN” is issued.
- ◆ The size of the TWA available to the target program is that given in the PCT for the MANTIS transaction.
- ◆ If the target program contains an “EXEC CICS RETURN”, control is not returned to the MANTIS program invoking it.

---

**/XCTL**

**Description**     *Required.* Indicates that you want to transfer control to the program.

**Format**            Must immediately follow the target program name

**Considerations**

- ◆ In CICS, if you restart MANTIS (with START or XCTL) it does *not* start at the next statement following the PERFORM. It starts initializing with MASTER\_SIGNON.
- ◆ If MANTIS has issued a DBMS SINON, no SINOF is issued with "PERFORM ... XCTL". This can cause dangling DBMS tasks to remain in the system, unless the performed program issues a SINOF.
- ◆ See also "CALL" on page 137.

**General considerations**

- ◆ The object of the PERFORM can be any text expression. The literal shown here as an example is "*program*/XCTL".
- ◆ Applies to CICS only.
- ◆ See also "CALL" on page 137, "DO" on page 206, and "RUN" on page 391.

**Example**            The following example shows the PERFORM statement transferring control to an external program:

```

00010 ENTRY FACILITY
00020 .SCREEN MAP( "CUSTOMER" )
00030 .CONVERSE MAP
00040 .IF KEY="PF12"
00050 ..PERFORM "EXTNPGM1/XCTL"
00060 .END
00070 .WHILE MAP<>"CANCEL"
      .
      .

```

(EXTNPGM1 will not  
return control to  
the MANTIS program.)

**PERFORM transfers control to another program and saves  
MANTIS context without a return**

---

**PERFORM "program/EXEC"**

---

---

***program***

**Description**     *Required.* Specifies the name of the target program you want to invoke.

**Format**             Must be a text expression from 1–8 characters in length and must not  
                             contain embedded blanks

**Consideration (CICS)**

- ◆ The program must be defined in the PPT (Program Properties Table) or CSD (CICS System Definition file) under RDO (Resource Definition Online) and exist in the DFHRPL Load/Core Image Library.

---

**/EXEC**

**Description**     *Required.* Indicates that you want to transfer control to the program.

**Format**            Must immediately follow the target program name

**General considerations**

- ◆ The object of the PERFORM can be any text expression. The literal shown here as an example is “*program/EXEC*”.
- ◆ This option applies to CICS TP monitors only.
- ◆ The size of the TWA available to the target program is the size given in the PCT for the MANTIS transaction.
- ◆ If the target program contains an “EXEC CICS RETURN”, control is not returned to the MANTIS program invoking it.
- ◆ To return to MANTIS at the statement following the PERFORM, enter code such as:

```
EXEC CICS START TRANSID (resX) TERMINAL (current terminal)
```

or:

```
EXEC CICS RETURN TRANSID (resX)
```

where *resX* is a valid MANTIS resume transaction ID. See your Master User for details.

- ◆ Under CICS, if MANTIS has been passed a COMMAREA by a previous task, MANTIS passes the COMMAREA to the target program.
- ◆ MANTIS rolls out to temporary storage and then performs a transfer of control. The MANTIS program can then be restarted.
- ◆ If you XCTL back to MANTIS, it reinitializes. It does NOT resume at the statement following the PERFORM.
- ◆ See also “CALL” on page 137, “DO” on page 206, and “RUN” on page 391.

### Example

The following example shows the PERFORM statement transferring control to an external program while saving the MANTIS context:

```
00010 ENTRY FACILITY
00020 .SCREEN MAP( "CUST_SETUP" )
00030 .CONVERSE MAP
00040 .WHILE MAP<>"CANCEL"
00050 ..WHEN OPTION=1 OR MAP="PF1"
00060 ...CHAIN "CUSTOMER_NAMES"
00070 ..WHEN OPTION=3 OR MAP="PF3"
00080 ...PERFORM "ORDER/EXEC"
00090 ..END
00100 .END
00110 EXIT
```



---

ORDER does not automatically return control to the MANTIS program; however, MANTIS context is saved and can be restarted.

---



---

## PERFORM starts a MANTIS program as a background task

---

**PERFORM "[*trans-id*]/BACK,*user-id,password,program*[;*text-string*]"**

---

---

### *trans-id*

<b>Description</b>	<i>Optional.</i> Specifies MANTIS transaction code. If a <i>transid</i> is not explicitly specified, a resume <i>transid</i> is assigned to the background task by MANTIS. See your Master User for details.
--------------------	--

---

### */BACK*

<b>Description</b>	<i>Required.</i> Invokes a background task.
--------------------	---

---

### *user-id*

<b>Description</b>	<i>Required.</i> Specifies a valid MANTIS user that you want signed on in the background task.
--------------------	--

<b>Format</b>	Must be a valid MANTIS user ID
---------------	--------------------------------

---

### *password*

<b>Description</b>	<i>Required.</i> Provides password for the specified user.
--------------------	--

<b>Format</b>	Must be the corresponding password for the user ID specified above
---------------	--

---

### *program*

<b>Description</b>	<i>Required.</i> Specifies the name of the MANTIS program you want to use as a background task.
--------------------	---

<b>Format</b>	Must be a valid MANTIS program name
---------------	-------------------------------------

<b>Consideration</b>	If the program resides in a user that is not the signed-on user, it must appear in the following format:
----------------------	--

`user-name:program-name`

Note that this format permits a total length of 33 characters.

**text-string**

<b>Description</b>	<i>Optional.</i> Specifies a text string that you want to pass to the background MANTIS program.
<b>Format</b>	Must be 1–100 characters in length (minus the length of the program name) and must also appear as an argument on the ENTRY statement on the background MANTIS program

**General considerations**

- ◆ The object of the PERFORM can be any text expression. The literal shown here as an example is “[*trans-id*]/BACK”.
- ◆ MANTIS issues a start for a new, non-terminal associated task, passing the name of the MANTIS program to execute. The originating MANTIS task continues.
- ◆ Your Master User may have modified the MASTER:SIGN\_ON program, so that the *user-id*, *password*, and *text-string* are also available to that program. See your Master User for details.
- ◆ Applies to CICS TP monitors only.
- ◆ This statement initiates a MANTIS program under the resume *transid* of the program making the PERFORM request. The invoked program runs as a background task (the task is not attached to a terminal). The MANTIS program that is invoked as well as any programs it calls are not attached to a terminal either. If you try to perform terminal I/O (such as an error message display), MANTIS writes the I/O to an external ESDS VSAM file and terminates the task. Printer terminal writes are allowed. The terminal ID of the background task is BACK\$MAN. CICS assigns the background task a unique task number.
- ◆ When the program running as a background task causes the background task to end, control is returned to the CICS system. The MANTIS program that invoked the background MANTIS program is unaware of the background task status.
- ◆ MANTIS does not check the *text-string* entry for correctness or content.
- ◆ An ESDS VSAM file (CSOL) must be set up to capture background information about the background task, or it abends at completion. See your Master User for details. (Refer to *MANTIS Administration, OS/390, VSE/ESA*, P39-5005.)

- ◆ To stop a MANTIS background program, do one of the following:
  - Sign on to a user with MASTER:TERMINATE facility program.
  - Chain to MASTER:TERMINATE at the completion of the invoked program.
  - Modify the facility program to chain to MASTER:TERMINATE if `TERMINAL="BACK$MAN"`.
- ◆ When the background task terminates, control passes to the user's facility program. If the facility program contains any terminal I/O, an error is written to the CSOL journal. (The program can send data to the printer.) Contact your Master User for suggestions.
- ◆ If you do not want control passed to a facility program, and you wish the task to terminate, chain to the MASTER:TERMINATE program.
- ◆ The following message types are written to the CSOL (log) file. For more information, refer to *MANTIS Administration, OS/390, VSE/ESA*, P39-5005.

Error code	Explanation
NUCEBTE	End MANTIS background task.
NUCFBME	Program error background task statement.
NUCSBTE	Start MANTIS background task.
NUCTBME	Invalid terminal request statement.

- ◆ When running a background task, MASTER:SIGN\_ON is passed the parameters from the PERFORM. Your Master User must ensure that the MASTER:SIGN\_ON is able to handle incoming parameters. Refer to *MANTIS Administration, OS/390, VSE/ESA*, P39-5005, for more information.
- ◆ When running a MANTIS background task, during an attempt to write a record to the CSOL file, it is possible to get a scheduled task abend. The following table lists error codes you can receive and their meanings.
- ◆ See also *CALL* on page 137, *DO* on page 206, and *RUN* on page 391.

Abend code	Explanation
BDUP	CICS returned a status of “DUPREC”. This CICS status return means that VSAM indicated that the current record being written has a key that is the same as a record already on file.
BERR	CICS returned some unexpected status not covered by any other ABEND code.
BILL	CICS returned a status code of “ILLOGIC”. This CICS status code means that VSAM indicated an error condition not covered by any other CICS status code.
BINV	CICS returned a status code of “INVREQ”. This CICS status code means that either an error was detected in the parameters coded on the write request, or the FCT entry was not coded to allow the write to occur.
BIOE	CICS returned a status code of “IOERR”. This CICS status code means that VSAM indicated a physical I/O error has occurred on the file.
BISC	CICS returned a status code of “ISCINVREQ”. This CICS status code means that a remote system indicated a failure that does not correspond to a known condition.
BLEN	CICS returned a status code of “LENGERR”. This CICS status code means that the length specified on the write request exceeds the maximum length for this file.
BNFL	CICS returned a status code of “DSIDERR”. This CICS status code means that the CSOL file is not defined in the FCT.
BNOP	CICS returned a status code of “NOT OPEN”. This CICS status code means that the file was not open.
BNOS	CICS returned a status code of “NOSPACE”. This CICS status code indicates that the CSOL file is full.

**Example**

The following example uses the PERFORM statement to start a MANTIS program as a background task:

```
00010 ENTRY FACILITY
00020 .SCREEN MAP( "CUSTOMER" )
00030 .CONVERSE MAP
00040 .WHEN KEY="PF12"
00050 ..PERFORM"/BACK,EXAMPLES,CASINO,JACKSON"
00060 ...|START PROGRAM JACKSON AS A BACKGROUND TASK FOR
00070 ...|USER EXAMPLES WHOSE PASSWORD IS CASINO.
00080 ...|
00090 .WHEN KEY="PF6"
00100 ..PERFORM"/BACK,USR1,PSW1,MASTER:PGM1;WRITE"
00110 ...|START MANTIS PROGRAM MASTER:PGM1 FOR USR1 WHOSE
00120 ...|PASSWORD IS PSW1. PASS THE DATA STRING "WRITE" TO
00130 ...|MASTER:PGM1.
00140 ...|
00150 .END
```

# PERFORM starts a non-MANTIS program as a background task

PERFORM "*trans-id*/EXTN[,*text-string*]"

## *trans-id*

<b>Description</b>	<i>Required.</i> Specifies the name of the external task to be initiated.
<b>Format</b>	Must be a valid non-MANTIS CICS transaction ID defined in the PCT or CSD under RDO

## /EXTN

<b>Description</b>	<i>Required.</i> Indicates an external task.
--------------------	--

## *text-string*

<b>Description</b>	<i>Optional.</i> Specifies a text string for MANTIS to pass to the external task that can be accessed with the EXEC CICS RETRIEVE command.
<b>Format</b>	Must be a string from 1–200 bytes in length and separated from the EXTN indicator with a comma

### General considerations

- ◆ The object of the PERFORM can be any text expression; the literal shown here as an example is "*trans-id*/EXTN".
- ◆ MANTIS issues a start for a non-MANTIS, non-terminal associated task. The originating MANTIS task continues.
- ◆ Applies to CICS TP monitors only.
- ◆ An EXEC CICS RETURN in the target program does not return to the MANTIS program that invoked it. The target program normally returns to CICS.
- ◆ Ensure that the external task initiated runs as a task not attached to the terminal. The contents of the text string are application dependent.
- ◆ You can access the text string in the target program using the EXEC CICS RETRIEVE command. See also "**CALL**" on page 137, "**DO**" on page 206, and "**RUN**" on page 391.

**Example**      The following example shows PERFORM starting a non-MANTIS program as a background task:

```
00010 ENTRY FACILITY
00020 .SCREEN MAP( "CUSTOMER" )
00030 .CONVERSE MAP
00040 .WHEN KEY="PF12"
00045 ..|50 INITIATES TASK TSKU WITH NO DATA
00050 ..PERFORM"TSKU/EXTN"
00060 .WHEN KEY="PF6"
00065 ..|70 INITIATES TASK TSKV WITH TEXT-STRING READ-ONLY
00070 ..PERFORM"TSKV/EXTN,READ-ONLY"
00080 .END
00090 .|RETURNS HERE IF TASK TSKU OR TSKV IS INITIATED
00100 .|ELSE RETURNS PROGRAM LINKAGE UNSUCCESSFUL MESSAGE
```

PI

The PI function returns the value of PI (3.14159265).

PI

General consideration

See also “ATN” on page 94, “COS” on page 164, “SIN” on page 403, and “TAN” on page 454.

**Example**      The following examples show how the PI function returns the value of pi:

Example	Results	Comments
PI	3.141592653589	
SIN(PI/4)	.7071067811865	
00080 X=30		
00090   COMPUTE SIN(X) where X is in degrees		
00100 DEGREES_TO_RADIANS=PI/180		
00110 Y=SIN(X*DEGREES_TO_RADIANS)		
00120 SHOW Y:WAIT		

## POINT

The POINT function returns a number identifying the position where the last string addition or subtraction occurs when MANTIS evaluates the text expression argument.

---

**POINT (*t1* ± *t2*)**

---

---

***t1*, *t2***

**Description**    *Required.* Specifies any valid text or DBCS expression.

**Consideration** With subtraction (*t1-t2*), you can use POINT to determine if and where one string exists within another. If *t2* does not exist in *t1*, zero is returned. Because 0 evaluates to FALSE, and any other value is true, POINT can be used in a logical expression (see the [examples](#)). The value returned can also be used in substringing operations. Normally, you will use subtraction, because addition simply returns SIZE(*t1*).

### General consideration

See also “[PAD](#)” on page 346 “[UNPAD](#)” on page 474, and [substringing](#) in “[Text data](#)” starting on page 50.



Examples

For these examples:

TEXT CUST:CUST="222-22-2222"

Example	Results	Comments
POINT(CUST-"2")	1 (TRUE)	First occurrence was at position 1. Can use the following, for example, to check for the existence of "2" at any position: IF POINT(CUST-"2")
POINT(CUST-"-")	4 (TRUE)	First occurrence was at position 4.
POINT(CUST-"3")	0 (FALSE)	String "3" does not exist in CUST.
NAME="Doe,John"		
NAME(POINT(NAME-"")+1)	"John"	
NAME(1,POINT(NAME-"")-1)	"Doe"	
SHIP_DATE="02/01/30"	TRUE	True, if SHIP_DATE has at least two occurrences of "/" in the value. There are repeated "-" operators.
POINT (SHIP_DATE-" / "- " / ")		

The following example shows how the POINT function is used to show substrings based on an occurrence of one string in another:

```
00010 ENTRY POINT_FUNCTION
00020 .TEXT NAME(30)
00030 .NAME="DOE,JOHN T."
00040 .COMMA_PT=POINT(NAME-","")
00050 .IF COMMA_PT
00060 . .NAME=NAME(COMMA_PT+1)+" "+NAME(1,COMMA_PT-1)
00070 .END
00080 EXIT
RUN
JOHN T.  DOE
```

- ◆ The following example shows how you can use the POINT function to check for a valid function key:

```
00010 IF NOT(POINT("PF1 PF2 PF3 PF4 PF5 ENTER CANCEL"-MAP))
00020 . MSG= "INVALID KEY PRESSED, MUST BE ENTER, CANCEL, or PF1 -PF5"
00030 END
```

## PRINTER (Function)

The PRINTER function returns the current assignment.

### PRINTER

#### Example

Example	Results	Comments
PRINTER	"L84A"	Current assigned printer ID.
PRINTER(1,2)	"L8"	Substringing permitted.

**Consideration** The initial printer assignment is specified in the USER profile. See your Master User for details.

## PRINTER (Statement)

The PRINTER statement assigns the printer device where MANTIS routes output. The PRINTER function returns the current assignment.

---

**PRINTER=*t***

---

*t*

**Description**     *Required.* Specifies a TP monitor identification for the printer where MANTIS routes output.

**Format**            A 1–8 character text expression

**Consideration**   If your CICS printer identification is less than four characters, you must pad it on the right with blanks.

### General considerations

- ◆ If an OUTPUT statement specifies routing to a printer and the current program does not contain a PRINTER statement, MANTIS routes output to the default printer device (for the signed on user).
- ◆ Printer destination can be changed by the installation exit. See your Master User for details.
- ◆ In Batch MANTIS under MVS, the printer assignment specifies the DDNAME.
- ◆ In Batch MANTIS, the printer assignment is *not* supported in VSE.
- ◆ Argument for the PRINTER statement (*t*) is translated into uppercase upon execution of your program.
- ◆ See also “**PASSWORD**” on page 349, “**TERMINAL**” on page 455, and “**USER**” on page 497.

**Examples**        The following examples, which are equivalent to each other, show how to use the PRINTER statement to route output.

- ◆ The following example routes output in one line:
 

```
00010 PRINTER="L84A"
```
- ◆ The following statement routes output in three lines:
 

```
00010 TEXT DEVICE(4)
00020 DEVICE="L84A"
00030 PRINTER=DEVICE
```

# PROGFREE

The PROGFREE function returns the number of bytes remaining in the program area.

## PROGFREE

### General considerations

- ◆ Use the PROGFREE function to determine the size of the current program.
- ◆ You can break up your program, or make it smaller (e.g., by removing comments) if there is little free program space left.
- ◆ See also “DATAFREE” on page 176 and “USERWORDS” on page 498.

### Example

The following example shows how the PROGRFREE function returns the number of bytes still available in the program area:

Example	Results	Comments
PROGFREE	65488	Value for an empty program.

---

## PROGRAM

The PROGRAM statement identifies an external subroutine to be invoked by a DO statement.

---

```
PROGRAM name1([library1:]program-name1,password1)
      [, name2([library2:]program-name2,password2) . . . ]
```

---

### *name*

- |                      |   |
|----------------------|---|
| <b>Description</b>   | <i>Required.</i> Specifies the name you use to refer to your program in subsequent DO statements. |
| <b>Format</b>        | A MANTIS symbolic name (see “ <a href="#">Symbolic names</a> ” on page 24)                        |
| <b>Consideration</b> | When the symbolic name is previously defined, MANTIS bypasses this definition.                    |

---

### *[library:]program-name*

- |                    |   |
|--------------------|---|
| <b>Description</b> | <i>Required.</i> Specifies the name of the program as you saved it in program design. |
| <b>Format</b>      | 1–49 character text expression that evaluates to a valid program name                 |

### Considerations

- ◆ If the subroutine is in another user’s library, you can access it by specifying the name of the user in whose library it does reside (*[library1:]*).
- ◆ If the subroutine resides in your library, you can specify only the program name.
- ◆ If this parameter is used, the colon (:) is required.
- ◆ If you want this entity to be HPO bound, the library name is required, even if it is your own library.

**password**

**Description**     *Required.* Specifies the password as you saved it during program design.

**Format**            1–16 character text expression that evaluates to a valid password

**General considerations**

- ◆ Execute the PROGRAM statement only when you are sure your program will execute an external subroutine. For unbound programs, defer executing PROGRAM statements as long as possible. If the external program is in the HPO Shared Pool, relative positioning of PROGRAM statements prior to the DO is not significant.
- ◆ You must place an ENTRY-EXIT statement pair around the top level routine in the subroutine you invoke. A program can DO both internal and external subroutines.
- ◆ To avoid overhead during reexecution, keep complex variable types (SCREEN, FILE, ACCESS, TOTAL, VIEW, and INTERFACE) in the highest-level routine possible and pass them as parameters.
- ◆ Do not make every routine external, but group related routines together in one program with multiple internal routines. You can use MANTIS COMPONENTS to keep a single version of source code in multiple programs
- ◆ Programs belonging to the CONTROL user are not available as external subroutines.
- ◆ The SLICE and SLOT statements are ignored in an externally done program.

- ◆ Binding a PROGRAM statement saves the library code (and name) of the external routine. If you transfer a bound program to another system, the same library name can have a different library code. Rebind any transferred programs containing bound PROGRAM statements on the target system. Your Master User can help you determine the library code.
- ◆ The *library* and *program-name* arguments for the PROGRAM statement are translated to uppercase upon execution of your program.
- ◆ See also “COMPONENT” on page 153, “DO” on page 206, “ENTRY-EXIT” on page 213, “RELEASE (Function)” on page 378, and “RELEASE (Statement)” on page 380.

### Example

The following example shows how the PROGRAM statement invokes an external subroutine:

```

00010 ENTRY EDIT_PROGRAM
00030 .TYPE="CREDIT CHECK"
00040 .PROGRAM EDIT_RTN("VALIDATION", "COMMON")
00050 .DO EDIT_RTN(TYPE,CUST_NO,STATUS,MESSAGE)
00060 .IF STATUS<>"GOOD"
00070 ..DO ERROR_RTN(CUST_NO)
00080 .END
00090 .TYPE="SELECT SALES REP"
00100 .DO EDIT_RTN(TYPE,CUST_NO,STATUS,MESSAGE)
00110 .IF STATUS<>"GOOD"
00120 ..DO ERROR_RTN(SALES_REP)
00130 .ELSE
00140 ..SALES_REP=MESSAGE
00150 .END
00160 EXIT
00170 ENTRY ERROR_RTN(FIELD)
00180 .IF NOTE=" "
00190 ..NOTE=MESSAGE
00200 ..ATTRIBUTE(MAP,FIELD)="BRI,CUR"
00210 .ELSE
00220 ..ATTRIBUTE(MAP,FIELD)="BRI"
00230 .END
00240 EXIT

```

# PROMPT

The PROMPT statement displays a prompter. MANTIS retrieves the prompter from the library and displays it. In the case of chained prompters, MANTIS displays each prompter in the chain. Following the PROMPT, MANTIS returns control to the next line in the program.

---

**PROMPT** [*library:*]*prompter-name*

---

---

**[*library:*]*prompter-name***

**Description**     *Required.* Specifies the name of an existing prompter.

**Format**            1–33 character text expression

**Considerations**

- ◆ If the prompter is in another user's library, you can access it by specifying the name of the user in whose library it does reside [*library:*]*prompter-name*.
- ◆ If the prompter resides in your library, you may specify only the *prompter-name*.
- ◆ This parameter is translated to uppercase upon execution of your program.



## General considerations

- ◆ Press ENTER to view the next screen in the prompter. When prompter information ends, MANTIS returns control to the next line in the program. You can terminate the PROMPT statement by pressing the CANCEL key. You can terminate the program running the PROMPT statement by entering the KILL keyword in the lower left corner of the screen displaying the prompter. The actual keyword (KILL) can be changed by the installation. See your Master User for details.
- ◆ Prompters are limited to 80 lines. You can chain prompters to expand their capacity. (Refer to the Prompter Design Facility in *MANTIS Facilities, OS/390, VSE/ESA*, P39-5001, for more information.)
- ◆ Because program comments consume space in a stored program and require run-time overhead, put large blocks of comments in a prompter. Use a PROMPT statement in immediate mode to view your comments.
- ◆ See also “KEY” on page 301.

## Example

The following example shows how the PROMPT statement retrieves and displays a prompter:

```

      .
      .
      .
00100 IF REQUEST="?"
00120 .PROMPT"MASTER:FACILITY_HELP"
00130 END
      .
      .
      .

```

# RELEASE (Function)

The RELEASE function returns a text string indicating the current release, environment, and copyright information about the MANTIS system that is executing.

## RELEASE

**Description** Returns a text string with current release information

**Format** *rrss.lll mm...m copyright . . .*

where:

Value	Description	Example
rr	Release	55
ss	Service Level	01
lll	Maintenance level	007
mm...m	Monitor ID	(10-bytes fixed length) CICS,MVS/BATCH, IMS/DC, IMS/BATCH VM/CMS, CMS/BATCH, TSO, UTM, TIAM/BATCH, AIM
copyright	Copyright notice	Legal description

## Examples

Example	Results	Comments
RELEASE	"5501.007 CICS MANTIS Copyright 1986, 1987, 1992, 1993, 1995,1997, 1998, 2001 Cincom Systems, Inc. All rights reserved."	
RELEASE(1,4)	"5501"	Substringing permitted.

- ◆ The following example shows how you can determine the RELEASE level of a running system:

```

EDIT L1 --- MASTER:MASTER                COLUMNS 1 73
COMMAND ==> SHOW RELEASE                   SCROLL ==> CUR
***** START OF PROGRAM *****
***** END OF PROGRAM *****

*****
*
* 5501.007 CICS      MANTIS Copyright 1986,1987,1992,1993, 1995,*
* 1997, 1998, 2001 Cincom Systems, Inc. All rights reserved.*
*
*
*
*
*****

```

## RELEASE (Statement)

The RELEASE statement frees RDM's internal storage for one specific view or for all views currently opened. It also frees internal storage for programs loaded with a PROGRAM statement.

---

```
RELEASE [ view – name  
         program – name ]
```

---

---

### *view-name*

**Description**     *Optional.* Specifies the name (as defined in a previously executed VIEW statement) of the logical view you want to release.

**Format**            A MANTIS symbolic name (see “[Symbolic names](#)” on page 24)

#### **Considerations**

- ◆ All views remain open until you sign off from MANTIS or until you perform an explicit RELEASE (with the RELEASEV=N customize option only).
- ◆ A RELEASE with a view-name releases only the named view. (See the first example.) A RELEASE without an argument releases all open RDM views.

---

### *program-name*

**Description**     *Optional.* Specifies the name (as defined in a previously executed PROGRAM statement) of an external subroutine.

**Format**            A MANTIS symbolic name (see “[Symbolic names](#)” on page 24)

#### **Considerations**

- ◆ RELEASE *program-name* frees the memory occupied by the program (loaded with a PROGRAM statement). (See the second example.)
- ◆ RELEASE removes the program from the local program chain.
- ◆ The next DO for *program-name* will reload the program.

## General considerations

- ◆ When you are accessing multiple logical views, use RELEASE to free the memory used by the views opened. MANTIS removes all MARKs and the current position is no longer known to RDM. However, if you execute GET, INSERT, UPDATE, or DELETE on a released view, RDM automatically opens that view. You cannot use RELEASE to assign new SELECT qualifications to the VIEW.
- ◆ When a RELEASE statement is issued against a program that is no longer in storage or is in the shared pool, MANTIS ignores this statement.
- ◆ See also “PROGRAM” on page 373 and “VIEW” on page 501.
- ◆ The RELEASE statement frees RDM’s internal storage for one specific view or for all views currently opened. It also frees internal storage for programs loaded with a PROGRAM statement.
- ◆ On a RELEASE *view-name*, TRAP *view* ON is in effect and returns a status of either GOOD or ERROR, as well as the FSI/VSI information that is returned on the RELEASE (see “TRAP” on page 469 and “Status functions” on page 517 for more details).

## Examples

- ◆ The following example shows how the RELEASE statement is used to free internal storage for one specific view:

```
00010 VIEW CUSTOMER( "CUST" )
00020 DO PROCESS_VIEW(CUSTOMER)
.
.
.
00060 RELEASE CUSTOMER
.
.
.
```

- ◆ The following example shows how the RELEASE statement is used to free internal storage for a program loaded with a PROGRAM statement:

```
00010 PROGRAM INFREQUENT( "OCCASIONAL" )
.
.
.
00050 DO INFREQUENT(A,B,C)
00060 RELEASE INFREQUENT
```

# REPLACE

The REPLACE statement identifies the library, program name, password, and description of the executable program that is replaced in your library as the result of issuing the Compose action on a MANTIS source program.



If UPPERCASE=N has been specified in the FSE (Full Screen Editor), you must enter this statement in UPPERCASE mode for it to be recognized by MANTIS.

**REPLACE**"[*library:*] *program-name* [/password] [/description]"

## *library:*

<b>Description</b>	<i>Optional.</i> Specifies the name of the library where the executable program is replaced.
<b>Default</b>	Your sign-on library
<b>Format</b>	A MANTIS symbolic name, 1–16 characters in length, followed by a colon (:) (see “ <a href="#">Symbolic names</a> ” on page 24)
<b>Considerations</b>	<ul style="list-style-type: none"> <li>◆ Must be your user library.</li> <li>◆ Code the REPLACE statement for your library only.</li> <li>◆ This parameter is translated to uppercase upon execution of your program.</li> </ul>

## *program-name*

<b>Description</b>	<i>Required.</i> Specifies the name of the executable program that is replaced in your library as a result of the Compose action.
<b>Format</b>	A MANTIS symbolic name, 1–32 characters in length (see “ <a href="#">Symbolic names</a> ” on page 24)
<b>Consideration</b>	This parameter is translated to uppercase upon execution of your program.

***/password***

<b>Description</b>	<i>Optional.</i> Specifies the password used to previously replace the executable program (or the password used to save the program for the first time).
<b>Default</b>	Your sign-on password
<b>Format</b>	Must be a standard password, 1–16 characters in length, preceded by a slash (/); can be upper or lowercase

---

***/description***

<b>Description</b>	<i>Optional.</i> Specifies the text description of the executable program.
<b>Format</b>	Text expression, 1–46 characters in length, preceded by a slash (/); can be upper or lowercase
<b>Consideration</b>	If you specify a description for the REPLACE statement, it becomes the description for the composed program when the Compose action is completed.



---

The description of the REPLACE statement is optional and should be used for user reference purposes. It is not used to create or update the description when the executable program is saved or replaced.

---



## General considerations

- ◆ The REPLACE statement is optional. If used, it cannot be continued from one line to the next. Code a single REPLACE statement on a single line of a MANTIS source program. The REPLACE statement cannot be coded before the ENTRY statement. The recommendation is to code the REPLACE statement following the first ENTRY statement.
- ◆ For more information on the REPLACE command, refer to *MANTIS Program Design and Editing, OS/390, VSE/ESA*, P39-5013.
- ◆ If you append your source program name with an at sign (@), the REPLACE statement is not needed. The at sign (@) is the system default that tells the Compose action to name the composed program with the same name as the source program without the at sign (@). The @ symbol is defined by the Master User and can be different for your installation.

For example, if you compose CUST\_BROWSE@, the resulting composed program is CUST\_BROWSE. If you do not append the at sign (@) to the source program, supply a REPLACE statement that specifies the name of the executable program to be replaced. A source program to be composed without the at sign (@) and without a REPLACE statement generates an error message.

- ◆ Double quotes (") are required around the parameters of the REPLACE statement. The colon (:) is required to separate library and program name, and the slash character (/) is required to separate password and description as shown.
- ◆ You can select the REPLACE statement in the Full Screen Editor with the S (select) line command for *n*-level editing. For more information about the S line command and the Compose and Decompose actions, refer to *MANTIS Program Design and Editing, OS/390, VSE/ESA*, P39-5013.

- ◆ The *library* and *program-name* arguments for the REPLACE statement are translated to uppercase upon execution of your program.
- ◆ The COMPOSE action will convert the REPLACE statement to a comment in the composed program.
- ◆ See also “**COMPONENT**” on page 153, “**CSIOPTNS**” on page 165, and “**REPLACE**” on page 383.

**Example**      The following example shows how the REPLACE statement identifies the program that has been replaced by issuing a COMPOSE action on a MANTIS source program:

```
00010 ENTRY CUST_INSERT
00020 REPLACE"ACCT:CUST_INSERT/DEPT1234/CUSTOMER RECORD INSERT PROGRAM"
00030 CSIOPTNS"COMMENTS=NO:FORCE=YES:SEQUENCE 5,5"
      .
      .
      .
00520 EXIT
00530 COMPONENT"ACCT:CUS_INIT_FILE_HEADER"
00540 COMPONENT"ACCT:CUS_ERROR_PROC"
00550 COMPONENT"ACCT:CUS_TERMINATE"
```

# RESET

The RESET statement backs out a Logical Unit of Work (LUW). MANTIS rolls back any updates made since the start of a Logical Unit Work. You can only back out updates when supported by the teleprocessing system and file system.

## RESET

### General consideration

See also “**COMMIT**” on page 149. For more information about the FSE COMMIT command, refer to *MANTIS Facilities, OS/390, VSE/ESA*, P39-5001.

**Example**      The following example shows the RESET statement backing out a LUW:

```

00007 WHILE MAINREC<>"END"
00008 .GET MAINREC
00009 .INSERT PARTIAL1
00010 .IF PARTIAL1="ERROR"
00011 ..SHOW"ERROR OCCURRED ON 1ST INSERT":WAIT
00012 .ELSE
00013 ..INSERT PARTIAL2
00014 ..IF PARTIAL2="ERROR"
00015 ...RESET
00016 ...SHOW"ERROR OCCURRED ON 2ND INSERT, 1ST INSERT BACKED-OUT":WAIT
00017 ..ELSE
00018 ...INSERT PARTIAL3
00019 ...IF PARTIAL3="ERROR"
00020 ....RESET
00021 ....SHOW"ERROR OCCURRED ON 3RD INSERT,1ST & 2ND INSERT BACKED-OUT":WAIT
00023 ...ELSE
00024 ....COMMIT
00025 ....SHOW"ALL THREE INSERTS SUCCESSFUL, PROCESSING NEXT RECORD":WAIT
00026 ...END
00027 ..END
00028 .END
00029 END

```

## RETURN

Use the RETURN statement to return control from a subroutine, or to stop execution of a program, before the physical end of the program (EXIT statement).

---

### RETURN

---

#### General considerations

- ◆ Use the RETURN statement when you want a subroutine to return control from any line within the ENTRY-EXIT statements. MANTIS returns control immediately as if the EXIT statement had been executed.
- ◆ RETURN in an internal subroutine will return to the statement after the DO that invoked it.
- ◆ If you use a RETURN statement in your program at dolevel 0, the effect is the same as a STOP statement. MANTIS returns to programming mode if the program was executing while in programming mode; or, MANTIS returns to your Facility Selection menu if the program was executing while not in programming mode.
- ◆ See also “BREAK” on page 136 and “NEXT” on page 335.

#### Example

The following example shows how the RETURN statement returns control from a subroutine:

```
10 ENTRY BROWSE
20 .SCREEN MAP1( "INDEX" )
30 .FILE REC1( "INDEX", "SERENDIPITY" )
40 .GET REC1
50 .WHILE REC1="NEXT"
60 ..CONVERSE MAP1
70 ..IF MAP1="CANCEL" <---- If this condition is TRUE, the RETURN
80 ...RETURN verb will continue logic flow to the
90 ..END EXIT statement (120)
100 ..GET REC1
110 .END
120 EXIT
```

---

## RND

The RND function returns a random real number in the range zero to  $a$ , but excluding zero and  $a$ .

---

**RND( $a$ )**

---

---

**$a$**

**Description**     *Required.* Specifies the ending range for the randomizer function.

**Format**            Any valid arithmetic expression

**Consideration**   Should be non-zero for meaningful results.

**General consideration**

See also “SEED” on page 398. Without the SEED statement, the random number generator produces the same sequence of numbers each time you execute a program. With the SEED statement, the internal system clock seeds the random number generator.

**Example**      The following example shows how the RND function returns random numbers:

Example	Results	Comments
RND(10)	.1330482773482E-2	
RND(1)	.1330482773482E-3	The series will always be the same unless you execute a SEED statement.
RND(1)	.2361423983238	
RND(1)	.8452904769219	
RND(1)	.7970522423274	
RND(1)	.5704303598031E-1	
RND(1)	.7223061672411	
SEED		SEED sets a random starting number.
RND(1)	.1898911846801	
RND(1)	.5011424054391	
RND(-1)	-.133048277E-3	
RND(-1)	-.236142398	
X    ROUNDED(0)=	1	
RND(1000)		
X    ROUNDED(0)=	236	
RND(1000)		
RND(0)	0	Exception case.

# RUN

The RUN command executes the program currently in the programming mode work area.

A program runs until one of the following occurs:

- ◆ The program encounters an error.
- ◆ The program encounters one of the following statements:
  - CHAIN
  - EXIT
  - RETURN
  - STOP
- ◆ The program runs out of statements.
- ◆ You issue a KILL command.

---

## RUN [*n*]

---

*n*

<b>Description</b>	<i>Optional.</i> Specifies the statement number where you want the run to begin.
<b>Default</b>	First line in your program
<b>Format</b>	Arithmetic expression that evaluates to a value 1– <i>n</i> , where <i>n</i> is the maximum line number in your program
<b>Consideration</b>	MANTIS uses only the integer portion of <i>n</i> .

## General considerations

- ◆ **RUN with no parameter.** MANTIS erases definitions and values for all symbolic names, except for ENTRY symbolic names, in the current program's data area and executes the program from the lowest numbered statement.
- ◆ **RUN *n*.** MANTIS retains all variables and arrays, as well as their current values, from prior RUN executions and executes the current program from the specified statement number. Use this form of the RUN command to continue a program after it has been halted by an error condition or by a STOP statement. This is useful for interactive testing and debugging.
- ◆ The RUN command is supported at the second level of edit. That is, a program that is being edited at the second level can be executed using the RUN command. RUN works the same in a second level edit as at the top level *except for* the four differences listed below:
  - If a second-level edit program containing an external DO also contains an error within the external DO program (or is KILLED), the current program replaces the second-level program being edited, and the edit level is unchanged.
  - If the second-level edit program contains a CHAIN, the chained-to program becomes the second-level program and any changes are lost.
  - Issuing a LOAD or NEW command automatically ends the second-level edit session. All changes are lost. The edit level of the new program will be at the top level. (MANTIS sends a warning message if you issue LOAD or NEW.)
  - A CANCEL at the second level causes MANTIS to return to the top level. The END, MENU, and LOGOFF commands (described in *MANTIS Program Design and Editing, OS/390, VSE/ESA*, P39-5013) also return MANTIS to the top level, saving any changes made at the second level.
- ◆ The program must be logically correct (balanced ENTRY-EXIT, WHILE-END, etc.) before you can RUN it.
- ◆ See also “STOP” on page 452.

## Example

The following example shows how the RUN statement is used to execute the program currently in the work area:

```
==> RUN
00010  SHOW"WHAT IS THE CAPITAL AMOUNT?"
      WHAT IS THE CAPITAL AMOUNT?
```



---

## SCREEN

The SCREEN statement specifies a screen design (screen) that you use in your program.

---

```
SCREEN name1([library1 :]screen – name1,[PREFIX])  
          [, name2([library2 :]screen – name2,[PREFIX]) ... ]
```

---

### *name*

**Description**     *Required.* Specifies the name used to refer to a screen in subsequent CONVERSE statements.

**Format**            A MANTIS symbolic name (see “[Symbolic names](#)” on page 24)

### **Considerations**

- ◆ When the symbolic name is previously defined, MANTIS bypasses this definition.
- ◆ After a CONVERSE, the symbolic name contains the mnemonic for the last key pressed.

## ***[library:]screen-name***

**Description**     *Required.* Specifies the name of the screen as saved during Screen Design.

**Format**            1–49 character text expression that evaluates to a valid Screen Design name

### **Considerations**

- ◆ If the screen is in another user's library, you can access it by specifying the name of the user, followed by a colon and the screen name, as follows:

*library:screen-name*

If the screen resides in your library, you can specify only the screen name.

- ◆ If you want this entity to be HPO bound, the library name is required, even if it is in your own library.
- ◆ This parameter is translated to uppercase upon execution of your program.
- ◆ If this parameter is used, the colon (:) is required.

## PREFIX

**Description** *Optional.* Specifies whether MANTIS places the symbolic name and an underscore before all field names associated with this screen. If you code:

```
10 SCREEN BIN( "BOTTLES" , PREFIX)
```

and the screen design BOTTLES had a field named VOLUME, the program would refer to that field now as BIN\_VOLUME.

**Consideration** See the PREFIX considerations for “**FILE**” starting on page 222.

### General considerations

- ◆ A SCREEN statement is bindable only when the user-name is provided, even if the screen resides in your own library.
- ◆ Arguments for the SCREEN statement are translated into uppercase upon execution of your program.
- ◆ See also “**CONVERSE**” on page 157.

**Example** The following example shows how the SCREEN statement specifies a screen design that is used:

```
00020 .FILE RECORD( "INDEX" , "SERENDIPITY" , 16)
00030 .SCREEN MAP( " INDEX" )
00040 .WHILE RECORD<> "END" AND MAP<> "CANCEL"
00050 ..CLEAR MAP:BUFFER=1
00070 ..GET RECORD( "WILLIAMS" ) LEVEL=BUFFER
00080 ..WHILE RECORD<> "END" AND BUFFER<17
00090 ...BUFFER=BUFFER+1
00100 ...GET RECORD LEVEL=BUFFER
00110 ..END
00120 ..CONVERSE MAP
00130 .END
```

---

# SCROLL

The SCROLL statement allows you to set the scrolling mode of the terminal or to specify (within your program) window-mode scrolling increments for PF keys.

---

```
SCROLL [ OFF
        ON
        [row][,col/] ]
```

---

---

## OFF

- Description**     *Optional.* Specifies that data displayed with a SHOW statement scrolls from top to bottom on the screen.
- Format**             Must be coded exactly as shown
- Consideration** Data displayed with SCROLL OFF overwrites the oldest line on the screen. The display starts at the top of the screen and the last line of output appears in high intensity. This can be used to minimize the amount of data transmitted to the terminal.

---

## ON

- Description**     *Optional.* Specifies that all data displayed with a SHOW statement appear at the bottom of the screen. All previous lines are bumped up by one line and the top line is lost.
- Format**             Must be coded exactly as shown
- Consideration** ON is the default value if no parameter is entered.

---

## row

- Description**     *Optional.* Specifies window-mode row increment value. The increment value is not altered if you supply a zero value or no data.
- Format**             Arithmetic expression that evaluates to a value in the range of 0 through 255. Mantis uses only the integer part of row.
- Default**             The number of displayed rows on the terminal.

---

**col**

- Description**     *Optional.* Specifies window-mode column increment value. The increment value is not altered if you supply a zero value or no data.
- Format**            Arithmetic expression that evaluates a value in the range of 0 through 255. Mantis uses only the integer part of *col*.
- Default**            The number of displayed columns on the terminal.

**General considerations**

- ◆ Scroll is automatically ON when the SCROLL statement is not used in the program. If you use the statement, you must supply one of the parameters.
- ◆ Use SCROLL *row,col* to establish meaningful scroll values for the map set you are about to converse. For example, you may want to scroll to a specific row or column.
- ◆ See also “TERMSIZE” on page 456.

**Example**            The following example shows how the SCROLL statement is used to set the scroll mode of the terminal:

```
00030 SHOW"PLEASE ENTER MINIMUM AND MAXIMUM RANGE "
00040 SHOW"OF NUMBERS TO BE SELECTED.
00050 SHOW"INPUT IN THE FORMAT OF MIN;MAX (EX. 999;999)"
00060 OBTAIN MIN,MAX
00070 CLEAR
00080 SCROLL OFF
00090 GET RECL
```

## SEED

The SEED statement seeds the random number generator so that it generates a new sequence of random numbers.

---

### SEED

---

#### General considerations

- ◆ Without the SEED statement, the random number generator produces the same sequence of numbers each time you execute a program. With the SEED statement, the internal system clock seeds the random number generator.
- ◆ See also “RND” on page 389.

#### Example

The following example shows how the SEED statement is used to set the random number generator:

```
00010 SEED
```

```
00020 A=10
```

```
00030 B=RND(A)
```

# SGN

The SGN function returns the algebraic sign of a numeric expression. SGN returns -1 if a is less than 0, 0 if a equals 0, and +1 if a is greater than 0.

**SGN(a)**

**a**

**Description**     *Required.* Specifies a numeric expression whose sign you want to determine.

**Format**            Specifies any valid numeric expression

**General consideration**

See also “[ABS](#)” on page 86.

**Examples**         The following examples show what the SGN function returns for various input values:

Example	Results	Comments
SGN(-14)	-1 (TRUE)	Any non-zero value is TRUE.
SGN(.00001)	+1 (TRUE)	
SGN(A*B)	0 (FALSE)	When A or B is zero, FALSE is returned.
SGN(1.4E-17)	+1 (TRUE)	

---

# SHOW

The SHOW statement displays and formats data on a screen. MANTIS outputs the specified data item(s) on the screen according to the scrolling method specified. MANTIS locates these data items on the line according to the AT, ',' and ';' options specified.

---

$$\text{SHOW } x1 \left[ \begin{Bmatrix} \{ \} \\ ; \\ \{ \} \end{Bmatrix} \right] x2 \left[ \begin{Bmatrix} \{ \} \\ ; \\ \{ \} \end{Bmatrix} \right] . . . x_n \left[ \begin{Bmatrix} \{ \} \\ ; \\ \{ \} \end{Bmatrix} \right]$$

$$\text{where each } x_n \text{ is: } \left[ \text{AT}(\text{tab}) \begin{Bmatrix} \{ \} \\ ; \\ \{ \} \end{Bmatrix} \text{ data - item} \right]$$

---

---

## AT (tab)

- Description**     *Optional.* Specifies the column where the data begins.
- Format**             Arithmetic expression that evaluates to a value in the range 1–*n*, where *n* is the maximum column number for your terminal
- Considerations**
- ◆ MANTIS uses only the integer portion of *tab*.
  - ◆ If you specify the AT clause, MANTIS outputs the data beginning at the character position specified by *tab*.
  - ◆ If an AT clause causes one data item to overlap an existing data item, or is less than the current column position, MANTIS skips to the next line with the second data item in the specified position, followed by any subsequent data items in the statement.

---

## data-item

- Description**     *Optional.* Specifies the data item(s) you want displayed.
- Format**             Arithmetic, text, or DBCS expression
- Consideration** You can specify different types of data items in a single SHOW statement.



## General considerations

- ◆ A SHOW statement with no parameter returns a blank line.
- ◆ If you enter a semicolon between data items, MANTIS returns one blank space between them.
- ◆ If you enter a comma between data items, MANTIS returns one item to a screen *zone*. Screen zones are 13 characters in size. The first zone starts in column 1, the next in column 14, then 27, and so on. MANTIS left-justifies data items and fills extra positions with spaces. If an item contains more than 13 characters, it continues into the next zone. The next data item begins in the following zone.
- ◆ MANTIS uses the entire terminal width to display the SHOW output.
- ◆ MANTIS displays significant digits for a numeric data item. Zero shows as a single blank. If needed, MANTIS displays a numeric value in E notation. If you want numeric data formatted a certain way, you can use the FORMAT function.
- ◆ If the SHOW statement ends with a data item, MANTIS terminates the line, and the next SHOW starts a new line of output. If the SHOW statement ends with a comma or semicolon, MANTIS continues the line with the next SHOW.
- ◆ If your SHOW statement ends with a semicolon (;), MANTIS displays the data in the Message Line at the next CONVERSE, WAIT, or OBTAIN. Subsequent SHOWs continue the line. See “[Enhanced screen and program design](#)” on page 555 for an illustration of the message line.
- ◆ To remove data that is already set by SHOW and is terminated by a semicolon (;), but has not yet been sent to a CONVERSE, either execute a CLEAR or a SHOW with a null line (SHOW “”).
- ◆ If you code a SHOW in an external program, and the CONVERSE is in the calling program, either code a SHOW before the DO statement in the calling program or have your Master User set the customization option ADAll=Y.
- ◆ If you use the semicolon with the SHOW (as a command) in Batch MANTIS, you must change the DELIMITER parameter from “;” to something else in your Batch MANTIS job. Refer to [MANTIS Facilities, OS/390, VSE/ESA](#), P39-5001, for information on using Batch MANTIS.

- ◆ When the screen fills up (for example, with 20 lines on a 24 line terminal), MANTIS sends the lines to the terminal. To avoid overwriting data, you must execute a WAIT or OBTAIN before issuing PROMPT or CONVERSE.
- ◆ The lines are also sent to the terminal when MANTIS executes a WAIT, OBTAIN, or STOP/KILL.
- ◆ The Printer Write and the Terminal Write Exits can affect this statement. See your Master User for details.
- ◆ MANTIS reserves four lines when SHOWs are used to build an unformatted display for the screen or printer. Two lines are reserved at the top of the output display for a heading (specified through the HEAD statement), and two lines are reserved at the bottom of the output display for messages and commands (e.g., KILL). On a model-2 terminal with 24 lines, a maximum of 20 lines can appear on an unformatted display generated via SHOWs. Formatted displays using screen design and CONVERSE statements are necessary to utilize the full number of lines available on the terminal or printer.
- ◆ There are three ways to obtain output in the Full Screen Editor from the following SHOW command:

```
SHOW x1, x2 . . .
```

- Where the result from x1, x2, . . ., is less than 54 characters, the output is displayed in the EDIT==> field.
  - Where the result is greater than 54 characters, the output is displayed in a window (5 rows by 66 columns) at the bottom right-hand side of the screen. MANTIS displays up to 330 characters in this window and truncates any additional characters.
  - A WAIT displays all SHOWs that have not been sent to the terminal (for statements in a running program).
- ◆ See also “CLEAR” on page 145, “FORMAT” on page 230, “OBTAIN” on page 341, “PRINTER (Function)” on page 370 (that determines how the lines display in your program), “PRINTER (Statement)” on page 371, “SCROLL” on page 396, and “WAIT” on page 506.

**Example**

The following example shows how the SHOW statement sets up data on an unformatted screen:

```
00010 FILE REC( "CUST_FILE",PASSWORD)
00020 HEAD"CUSTOMER LIST"
00030 GET REC
00040 WHILE REC<>"END"
00050 .SHOW NUMBER, LAST_NAME, FIRST_NAME
00060 .GET REC
00070 END
```

---

**SIN**

The SIN function returns the sine of *a* where *a* is in radians.

---

**SIN(*a*)**

---

***a***

**Description**     *Required.* Specifies the angle whose sine is to be returned.

**Format**            Any valid arithmetic expression

**Consideration** If the angle is in degrees, it must be converted to radians. See “PI” on page 367 for how to do this.

**General consideration**

See also “ATN” on page 94, “COS” on page 164, “SIN” on page 403, and “TAN” on page 454.

**Example**

The following example shows how the SIN function returns the sine of an arithmetic expression:

Example	Results	Comments
SIN(100)	-.506365641	
SIN(0)	0	
SIN(PI/4)	.7071067811865	

# SIZE

The SIZE function returns the maximum or current length of a field or the size of a text or DBCS expression. It can also return the number of defined dimensions for a field or array, as well as the number of occurrences for a specific dimension of an array.

SIZE (field – name

[ , "MAX"  
, "DIM"  
, n  
, "BYTlength" ]

)

## SIZE (t)

### field-name

- Description**     *Required.* Specifies the name of a field or an array. Datatype is TEXT, KANJI, BIG, or SMALL. This also works on symbolic names that return text status (INTERFACE, SCREEN, FILE, ACCESS, TOTAL, and VIEW).
- Consideration**   Must be a standard MANTIS symbolic name.

### t

- Description**     *Required.* Specifies a text or DBCS expression whose size you want to determine.
- Format**             A valid text or DBCS expression
- Consideration**   None of the second operands may be used with this format.

---

**"MAX"**

**Description**     *Optional.* Returns the maximum length (in characters) of a text or DBCS field; can be entered in either lower or uppercase.

**Consideration** Invalid for numeric fields.

---

**"DIM"**

**Description**     *Optional.* Returns the number of defined dimensions for a field; can be entered in either lower or uppercase.

---

***n***

**Description**     *Optional.* Returns the number of occurrences for the *n*th dimension of a field.

**Format**            An arithmetic expression evaluating to 1 or 2

**"BYTlength"**

**Description**     *Optional.* Returns the current bytlength of the specified string variable (including shift codes).

**Considerations**

- ◆ Invalid for numeric fields.
- ◆ Arrays must have an occurrence number.
- ◆ Bytlength can be abbreviated BYT.
- ◆ The results are the same whether or not bytlength is specified if the string field contains only single byte characters (that is, no Shift-in and Shift-out and/or double byte characters).

**General considerations**

- ◆ If you omit the second operand, the value returned is the current size of the string expression (*t* or *field-name*), in characters.
- ◆ If you specify the *n* parameter and the *n*th dimension of the field does not exist, MANTIS returns a value of zero.
- ◆ Fields are 2-dimensional if the second dimension is greater than 1, regardless of whether the first dimension is one. That is, MANTIS treats BIG VARIABLE (10,1) as equivalent to BIG VARIABLE (10). The following table provides some examples:

	(variable,"DIM")	(variable,1)	variable,2)
BIG VARIABLE	0	0	0
BIG VARIABLE(10)	1	10	0
BIG VARIABLE(1,10)	2	1	10
BIG VARIABLE(5,20)	2	5	20
TEXT VARIABLE(3)	0	0	0
TEXT VARIABLE(10,30)	1	10	0

- ◆ This function can be used to set level values that correspond to screen repeated fields or as limits when processing an array/matrix. It is also useful for dissimilarity debugging for dimensions (see “[Dissimilarity debugging](#)” on page 513 and the third and fourth examples below).
- ◆ MANTIS truncates trailing blanks for text fields when reading in from external files (ACCESS, INTERFACE, TOTAL, and VIEW), and MANTIS adds trailing blanks when retrieving external files.
- ◆ Arguments for the SIZE statement are translated to uppercase upon execution of your program.
- ◆ Refer to *MANTIS Facilities, OS/390, VSE/ESA*, P39-5001, for information on creating fields of 255 repeats.

## Examples

The following examples show the SIZE function:

```
00010 BIG A
00020 SMALL B(6),C(4,5)
00030 TEXT D(10),F(4,200)
00040 D="CANCEL"
00050 F(1)="PF3"
00060 KANJI K(8)
00070 PAD K
```

Example	Results	Comments
SIZE(D)	5	Current length
SIZE(F(1))	3	Current length
SIZE(F(2))	0	NULL
SIZE(D,"MAX")	10	
SIZE(C,"DIM")	1	
SIZE(F,"DIM")	2	
SIZE(B,1)	6	
SIZE(C,1)	4	
SIZE(C,2)	5	
SIZE(F,1)	4	
SIZE(K,"BYT")	16	

- ◆ The following example returns the number of repeats allowed on a terminal (when a repeat factor of 255 is specified, see Screen Design in *MANTIS Facilities, OS/390, VSE/ESA*, P39-5001). Assume FIELD A is defined on SCREEN1 with 255 vertical repeats. Also assume the same field is defined in FILE1.

```
00010 SCREEN MAP("SCREEN1")
00020 LIMIT=SIZE(FIELD A,1)
00030 FILE REC("FILE1","PASSWORD",LIMIT)
00040 I=1
00050 GET REC LEVEL=I
00060 WHILE I<LIMIT AND REC<>"END"
00070 .I=I+1
00080 .GET REC LEVEL=I
00090 END
```

*Limit now contains the number of repeats possible on the terminal where the application is now running.*

The defined map is now filled to the maximum possible number of repeats for that terminal.

- ◆ The following example shows how to use SIZE in dimension dissimilarity debugging. For example, when you receive the error message regarding dissimilar dimensions, you can determine the existing dimensions of the variable NAME.

If NAME is numeric (BIG or SMALL), enter:

```
SHOW SIZE(NAME,"DIM"),SIZE(NAME,1)
```

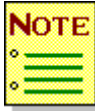
The previous statement shows the dimensions and maximum subscript value currently defined for NAME. If NAME is TEXT or DBCS/KANJI, enter:

```
SHOW SIZE(NAME,"DIM"),SIZE(NAME,1),SIZE(NAME,"MAX")
```

The previous statement shows the dimensions, the number of rows (first dimension), and maximum length currently defined for NAME.



- ◆ The following examples show how “BYT” returns the current bytelength of the specified string:



In the following examples, < means SO (Shift-out), and > means SI (Shift-in).

```
00010 MIXMODE ON
00020 TEXT ALPHA(20),BETA(20)
00030 KANJI GAMMA(20)
00040 ALPHA="< 1 2 >A"
00050 BETA="A<>B"
00060 GAMMA=G"< 1 2 >"
00070 A=SIZE(ALPHA,"BYT")
00080 B=SIZE(BETA,"BYT")
00090 C=SIZE(GAMMA,"BYT")
A, B, and C have values 7, 4, and 6.
```

If a second parameter is not specified, the current number of characters is returned. In the previous example, if you change the values to:

```
00070 A=SIZE(ALPHA)
00080 B=SIZE(BETA)
00090 C=SIZE(GAMMA)
```

A, B, and C have values of 3, 2, 2.

## SLICE

The SLICE statement limits the number of statements you can execute before MANTIS suspends your program.

---

**SLICE**  $\left\{ \begin{array}{c} n \\ \text{CLEAR} \end{array} \right\}$

---

---

***n***

**Description** *Required.* Specifies the number of statements a program can execute before MANTIS suspends it (the SLICE limit). In some environments, MANTIS suspends program execution and control returns to the TP monitor. Other environments do not support program suspension. In these circumstances, the program continues until the SLOT limit is reached.

**Format** Arithmetic expression

**Consideration** MANTIS uses only the integer portion of *n*. *n* must be 1–32767, inclusive.

---

## CLEAR

**Description** Resets the SLICE counter to 0.

### General considerations

- ◆ Every time a program executes, it can execute only the number of statements that SLICE specifies before MANTIS suspends it to give other programs and users an opportunity to run. When the program begins execution again, MANTIS provides another “SLICE” number of statements. This prevents the MANTIS application from monopolizing system resources.
- ◆ The default SLICE value is in the User Profile. You may wish to omit SLICE statements from your programs and tune the value in the User Profile for all programs.

- ◆ Program suspension is not supported or applicable in all environments. Some TP Monitors (such as CICS) provide this support, and MANTIS suspends program execution in these environments each time a “SLICE” number of statements has been executed. Other environments (IMS and Batch) do not support program suspension. In these environments, the SLICE value is reset, and the program continues to execute.
- ◆ If you omit the SLICE statement from a program, MANTIS assumes the value specified by your Master User in your User Profile.
- ◆ Note that specifying a large number in this statement or clearing SLICE from within your program can adversely affect performance of the system.
- ◆ MANTIS I/O statements have a slice value of 20 because of their relatively high demand on system resources. These statements are: GET, UPDATE, INSERT, DELETE, CALL, and EXEC\_SQL.
- ◆ If you set SLICE and SLOT equal to one, MANTIS executes your program one statement at a time.
- ◆ The SLICE statement cannot be set at any DOLEVEL other than 0. It must be large enough to accommodate printing large reports because when OUTPUT PRINTER is used, CONVERSE statements do not reset SLICE and SLOT.
- ◆ The product of SLICE and SLOT determines the number of statements your program can execute before it receives a program loop message.
- ◆ SLICE counter is reset to 0 at terminal I/O or at SLICE CLEAR.
- ◆ SLICE could be required for programs that execute a large number of statements between terminal I/O.
- ◆ Setting SLICE and SLOT to large numbers can adversely impact the performance of your overall online system or can lead to CICS task ABENDS instead of recoverable MANTIS Potential Program Loop messages.
- ◆ See also “COMMIT” on page 149 and “SLOT” on page 413.

### Example

The following example shows the SLICE statement setting a limit of program statements that can be reached before MANTIS suspends the program:

```
00032 ENTRY HOUSEKEEPING
00033 .FILE MAINREC("FILE1",PASSWORD1)
00034 .FILE PARTIAL1("PARTIAL1",PASSWORD2)
00035 .FILE PARTIAL2("PARTIAL2",PASSWORD3)
00036 .FILE PARTIAL3("PARTIAL3", PASSWORD4)
00037 .TRAP PARTIAL1 ON
00038 .TRAP PARTIAL2 ON
00039 .TRAP PARTIAL3 ON
00040 .SLICE 1000
00041 .SLOT 20
00042 EXIT
```

## SLOT

The SLOT statement specifies how many times a program can reach the SLICE limit before MANTIS returns “POTENTIAL PROGRAM LOOP ENCOUNTERED”. When you receive this message, you can press ENTER to continue running the program or use the KILL command to terminate execution.

---

**SLOT**  $\left[ \begin{array}{c} n \\ \text{CLEAR} \end{array} \right]$

---

*n*

- Description**     *Required.* Specifies the number of times a program can reach the SLICE limit before MANTIS returns a potential program loop error message.
- Format**            Arithmetic expression between 0–32767, inclusive
- Consideration**   MANTIS uses only the integer portion of *n*.
- 

## CLEAR

- Description**     Resets the SLOT counter to 0.

## General considerations

- ◆ When the number of SLOT counts is reached, MANTIS gives you a chance to break into a program that might be in an endless loop.
- ◆ The default SLOT value is in the User Profile. You may wish to omit SLOT statements from your programs and tune the value in the User Profile, which affects all programs.
- ◆ MANTIS resets SLICE and SLOT counters to zero every time a terminal output is executed on a screen. Output to a printer does not reset the SLICE and SLOT counters.
- ◆ When SLOT limit is set to 0, the SLICE value is not checked and no potential program loop error messages are issued when running under Batch MANTIS or CICS background. Setting SLOT to 0 can be used to allow long running jobs to execute in batch mode. TIME limit can be specified by JCL or shop standards to control looping tasks. If SLOT is set to 0 for an online program, it is equivalent to SLOT 1.
- ◆ The SLOT statement is ignored in an externally done program.
- ◆ If you set SLOT and SLICE limit equal to one, MANTIS executes your program one statement at a time.
- ◆ The SLOT statement cannot be set on any DOLEVEL other than 0. It must be large enough to accommodate printing large reports because when OUTPUT PRINTER is used, CONVERSE statements no longer reset SLICE and SLOT.
- ◆ See also “**COMMIT**” on page 149 and “**SLICE**” on page 410.

## Example

The following example shows the SLOT statement setting a limit of program statements that can be reached before MANTIS issues a potential loop warning message:

```
00032 ENTRY HOUSEKEEPING
00033 .FILE MAINREC("FILE1",PASSWORD1)
00034 .FILE PARTIAL1("PARTIAL1",PASSWORD2)
00035 .FILE PARTIAL2("PARTIAL2",PASSWORD3)
00036 .FILE PARTIAL3("PARTIAL3",PASSWORD4)
00037 .TRAP PARTIAL1 ON
00038 .TRAP PARTIAL2 ON
00039 .TRAP PARTIAL3 ON
00040 .SLICE 1000
00041 .SLOT 20
00042 EXIT
```

---

## SMALL

The SMALL statement names and gives dimensions to numeric variables. MANTIS creates a 4-byte numeric floating-point field or an array of 4-byte fields and associates it with the name you specify.

---

```
SMALL name1[(n1[,n2])]  
          [,name2[(n1[,n2])] . . . ]
```

---

### *name*

<b>Description</b>	<i>Required.</i> Specifies the name of the numeric variable.
<b>Format</b>	A MANTIS symbolic name (see “ <a href="#">Symbolic names</a> ” on page 24)
<b>Consideration</b>	When the symbolic name is previously defined, MANTIS bypasses this definition.

---

### *n1*

<b>Description</b>	<i>Optional.</i> Specifies the number of elements in a 1-dimensional array, or the number of rows in a 2-dimensional array.
<b>Default</b>	1
<b>Format</b>	Arithmetic expression that evaluates to a positive integer in the range 1–255
<b>Consideration</b>	MANTIS rounds <i>n</i> to an integer value.

---

### *n2*

<b>Description</b>	<i>Optional.</i> Specifies the number of columns in a 2-dimensional array.
<b>Default</b>	1
<b>Format</b>	Arithmetic expression that evaluates to a positive integer in the range 1–255
<b>Consideration</b>	MANTIS rounds <i>n</i> to an integer value.

## General considerations

- ◆ A SMALL variable contains a zero, upon initial definition.
- ◆ If  $n2$  is not specified, or is specified as 1, a 1-dimensional array is allocated. If  $n$  is not specified, a small scalar variable is allocated.
- ◆ Use BIG (instead of SMALL) to hold numbers involving fractions or more than 6 integer digits.
- ◆ You can use up to 2048 variables in each MANTIS program.
- ◆ See also “BIG” on page 134, “KANJI (Kanji users only)” on page 298, “TEXT” on page 457, and “Numeric data” on page 41.

## Examples

The following examples show how the SMALL statement names and gives dimensions to numeric variables.

```
00010 X=15
```

```
00020 SMALL ALPHA(64,3),BETA(12) Statements 20 and 30 are equivalent.
```

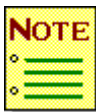
```
00030 SMALL ALPHA(8**2,SQR(9)),BETA(X-3)
```



---

## SOURCE

The SOURCE statement is coded in an executable program to name the library, program, password, and description of the source program to be created or replaced in your library by the Decompose action.



If UPPERCASE=N has been specified in the FSE (Full Screen Editor), you must enter this statement in UPPERCASE mode for it to be recognized by MANTIS.

---

**SOURCE**"*[library:] program-name [/password] [/description]*"

---

### *library:*

<b>Description</b>	<i>Optional.</i> Specifies the name of the library where the source program is created or replaced.
<b>Default</b>	Your sign-on library.
<b>Format</b>	A MANTIS library name, 1–16 characters in length, followed by a colon (:) (see “ <a href="#">Symbolic names</a> ” on page 24)
<b>Consideration</b>	Must be the current signed on library.

---

### *program-name*

<b>Description</b>	<i>Required.</i> Specifies the name of the source program that the Decompose action updates with the same changes that you made to the executable (composed) program.
<b>Format</b>	A MANTIS program name, 1–32 characters in length, (see “ <a href="#">Symbolic names</a> ” on page 24)

---

### */password*

<b>Description</b>	<i>Optional.</i> Specifies the password used to replace the source program previously (or the password used to save the program for the first time).
<b>Default</b>	Your sign-on password.
<b>Format</b>	A MANTIS symbolic name, 1–16 characters in length, preceded by a slash (/), can be entered in lower or uppercase (see “ <a href="#">Symbolic names</a> ” on page 24)

**/description**

- Description

Optional. Specifies the text description of the source program.
- Format

Text expression, 1–46 characters in length, preceded by a slash (/), can be entered in lower or uppercase
- Consideration

If you specify a description in the SOURCE statement, it becomes the description of the source program when the Decompose is issued.



The description of the SOURCE statement is optional and is used for user reference purposes. It is not used to create or update the description when the source program is saved or replaced.

**General considerations**

- ◆ Code the SOURCE statement for your library only.
- ◆ Code a single SOURCE statement in a MANTIS composed program when MANTIS source code changes. The recommendation is to code the SOURCE statement following the first ENTRY statement. If the SOURCE statement is not included in the composed program, and you change source code, the source program is not updated with the changes.
- ◆ The SOURCE statement must be nominated to be recognized by the Decompose action. Nominate a SOURCE statement by coding the at sign (@) character, (or another installation defined character) following the vertical bar ( | ) in the SOURCE statement, for example |@SOURCE.
- ◆ The SOURCE statement is required as shown below if you change MANTIS source code in the composed program and you want to keep those source changes when you issue the Decompose.

MANTIS source changes?	Component changes?	@SOURCE statement required?
Yes	No	Yes
No	Yes	No
No	No	No
Yes	Yes	Yes

- ◆ Double quotes (") are required around the SOURCE statement as shown in the previous example. The colon (:) is required to separate library and program name, and the slash character (/) is required to separate password and description as shown.
- ◆ The SOURCE statement cannot be continued from one line to the next. Be sure the statement is coded completely on a single line in the source program.
- ◆ You can select the SOURCE statement in the Full Screen Editor with the S (select) line command for *n*-level editing. For more information about the S line command, refer to *MANTIS Program Design and Editing, OS/390, VSE/ESA*, P39-5013.
- ◆ For more information about the use of the SOURCE statement and the Compose and Decompose actions, refer to *MANTIS Program Design and Editing, OS/390, VSE/ESA*, P39-5013.
- ◆ See also "COMPONENT" on page 153, "CSIOPTNS" on page 165, and "REPLACE" on page 383.

## Examples

- ◆ The following examples show the COMPONENT statement as it appears in a source program, a composed (executable) program, and nominated in a composed (executable) program for the Decompose action.

Example	Comments
00010 SOURCE"ACCT:CUST_ERROR_PROC"	Before Compose.
00010  *SOURCE"ACCT:CUST_ERROR_PROC"	Composed (executable) program after Compose.
00010  @SOURCE"ACCT:CUST_ERROR_PROC"	Composed (executable) program, component nominated for update in Decompose.

- ◆ The following example shows how the SOURCE statement names the executable program that is created or replaced by the Decompose action:

```

00010 ENTRY CUST_INSERT
00020 |@SOURCE"ACCT:CUST_INSERT@/DEPT1234/CUSTOMER RECORD INSERT - SOURCE"
.
.
.
00520 EXIT

```

---

# SQLCA (Function)

SQLCA is both a statement and a function. The SQLCA built-in function, shown below, returns data from the SQL Communication Area (SQLCA). The SQLCA statement stores data from the MANTIS program into the SQL Communication Area (SQLCA).

---

**SQLCA(*sqlca\_element\_name*)**

---

---

## *sqlca\_element\_name*

- Description**     *Required.* Specifies the element of the SQLCA that is to be transferred.
- Format**             A text expression that evaluates to one of the SQLCA element names in the following tables
- Consideration**    Element names must be selected from the list for the SQL database in use. See SQLCA elements in the “DB2” table on page 422 or SQLCA elements in the “SUPRA” table on page 423.

## General considerations

- ◆ Some SQLCA elements are not present in the native SQL SQLCA. These are extensions to the SQLCA unique to MANTIS. They are DBTYPE, DBNAME, MSGTEXT, and SQLISL. DBTYPE returns the SQL database currently in use (“DB2”, “SQL/DS” or “SUPRA”) to the MANTIS program. DBNAME returns the name of the SUPRA database currently in use. MSGTEXT returns the SQL error message text associated with the current SQLCA SQLCODE. SQLISL allows the DB2 for VSE and VM (formerly SQL/DS) ISOLATION parameter to be set by the MANTIS program.
- ◆ If an SQLCA TEXT element is moved to a MANTIS variable of shorter length (e.g., an 8-character SQLCA element to a 6-character MANTIS variable) the right-most characters are truncated.
- ◆ For further information, refer to *MANTIS DB2 Programming, OS/390, VSE/ESA*, P39-5028, or *MANTIS SUPRA SQL Programming, OS/390, VSE/ESA*, P39-3105.
- ◆ See also “EXEC\_SQL-END” on page 217, “SQLDA (Function)” on page 426, and “SQLDA (Statement)” on page 432.

## Example

The following example shows how data is retrieved from the SQLCA by the SQLCA built-in function. Line 160 checks the SQLCA SQLCODE to determine if all table rows have been fetched.

```
00130 EXEC_SQL
00140 .| FETCH C1 INTO :EMPL_NAME, :EMPL_NAME
00150 END
00160 IF SQLCA("SQLCODE")=100
00170 .DO END_OF_DATA
00180 END
```

The following table lists the SQLCA elements in DB2:

Element name	MANTIS compatible datatype	Contents / considerations	Updateable?
SQLCAID	TEXT(8)	Eyecatcher. Set by SQL.	No
SQLCABC	BIG	Length of SQLCA. Set by SQL.	No
SQLCODE	BIG	Code indicating the results of SQL statement execution.	Yes
SQLERRM	TEXT(70)	Tokens for insertion into SQL error message text. The vertical bar ( ) replaces hexadecimal "FF" as the separator character.	Yes
SQLERRP	TEXT(8)	SQL diagnostic data.	Yes
SQLERRD $n$	BIG	SQL diagnostic data. $n$ ranges between 1–6.	Yes
SQLWARN $n$	TEXT(1)	SQL warning flags. In DB2, $n$ ranges from 0–7. In DB2 for VSE and VM (formerly SQL/DS), $n$ ranges from 0–A.	Yes
SQLEXT	TEXT(8) DB2 TEXT(5) SQL/DS	Reserved for SQL.	Yes
DBTYPE*	TEXT(6)	Returns SQL database in use. Unique to MANTIS.	No
MSGTEXT*	TEXT(254)	Returns SQL error message text. Unique to MANTIS	No
SQLISL*	TEXT(1)	DB2 for VSE and VM (formerly SQL/DS) variable. Specifies ISOLATION parameter for Access Modules. Not supported by DB2.	Yes

\* These elements are MANTIS extensions to the SQLCA. They are not present in the SQL SQLCA.

The following table lists SQLCA elements in SUPRA:

Element name	MANTIS compatible datatype	Contents / considerations	Updateable?
SQLCAID	TEXT(8)	Eyecatcher. Set by SQL.	No
SQLCABC	BIG	Length of SQLCA. Set by SQL	No
SQLCODE	BIG	Code indicating the results of SQL statement execution.	Yes
SQLERRML	BIG	Length of SQL error message text.	No
SQLERRMC	TEXT(70)	SQL error message text.	Yes
SQLERRP	TEXT(8)	SQL diagnostic data.	Yes
SQLERRD $n$	BIG	SQL diagnostic data. $n$ ranges from 1–6.	Yes
SQLWARN $n$	TEXT(1)	SQL warning flags. $n$ ranges from 0–F.	Yes
DBTYPE*	TEXT(6)	Returns SQL database in use.	No
DBNAME*	TEXT(64)	Returns or sets SUPRA database name.	Yes

\*These elements are MANTIS extensions to the SQLCA. They are not present in the SQL SQLCA.

---

# SQLCA (Statement)

The SQLCA statement stores data from the MANTIS program into the SQL Communication Area (SQLCA). The SQLCA built-in function transfers data from the SQLCA to the MANTIS program.

The SQLCA statement is shown below.

---

**SQLCA(sqlca\_element\_name) = expression**

---

---

## sqlca\_element\_name

- Description**     *Required.* Specifies the element of the SQLCA that is to receive data.
- Format**             A expression that evaluates to one of the SQLCA element names in the preceding tables.
- Consideration**   Element names must be selected from the list for the SQL database in use. See SQLCA elements in the “DB2” table on page 422 or SQLCA elements in the “SUPRA” table on page 423.

---

## expression:

- Description**     *Required.* Specifies the data to be transferred into the SQLCA.
- Format**             Must be consistent with the datatype of the SQLCA element, text or numeric
- Consideration**   Certain SQLCA elements are read-only and cannot have data stored into them by the MANTIS program.



## General considerations

- ◆ Some SQLCA elements are not present in the SQL SQLCA. These are extensions to the SQLCA unique to MANTIS. They are: DBTYPE, DBNAME, MSGTEXT, and SQLISL. DBTYPE returns the SQL database currently in use ("DB2", "SQL/DS", or "SUPRA") to the MANTIS program. DBNAME is used to retrieve or set the SUPRA database value. MSGTEXT retrieves the SQL error message text associated with the current SQLCA SQLCODE. SQLISL retrieves or sets the DB2 for VSE and VM (formerly SQL/DS) ISOLATION parameter for the DB2 for VSE and VM Access Module.
- ◆ Although data can be stored in some SQLCA elements, doing so does not pass any information to the SQL database. The SQLCA is returned to the MANTIS program after each SQL statement is executed (EXEC\_SQL-END). Any data stored in the SQLCA will be overwritten when the next SQL statement is executed.
- ◆ For further information, refer to *MANTIS DB2 Programming, OS/390, VSE/ESA*, P39-5028, or *MANTIS SUPRA SQL Programming, OS/390, VSE/ESA*, P39-3105.

## Example

The following example shows how SQL error message text for an SQLCA error message can be retrieved by a MANTIS program. The SQLCA statement is used to store the SQLCODE value in the SQLCA and the MSGTEXT function is used to retrieve the error text. Note that the MSGTEXT function is not supported by SUPRA.

```
00150 TEXT SQL_ERROR_TEXT(254)
00160 SQLCA("SQLCODE")=(-504)
00170 SQL_ERROR_TEXT=SQLCA("MSGTEXT")
```

---

# SQLDA (Function)

SQLDA is both a statement and a function. The SQLDA statement stores data from the MANTIS program into the SQL Descriptor Area (SQLDA). The SQLDA function transfers data from the SQLDA into the MANTIS program.

## Read header elements

---

**SQLDA**(*sqlda\_name*, *sqlda\_header\_element*)

---

---

### *sqlda\_name*

- Description**     *Required.* Specifies the name of the SQLDA to be accessed.
- Format**            A text expression of 1–18 characters
- Consideration**   Naming conventions for SQL entities must be followed. Refer to the appropriate SQL language manual for the SQL database in use.

---

### *sqlda\_header\_element*

- Description**     *Required.* Specifies the SQLDA header element that is accessed.
- Format**            A text expression that evaluates to one of the SQLDA header element names shown in the “DB2” table on page 442 or the “SUPRA” table on page 443.

**Examples**

In the following examples, line 160 shows the SQLDA "SQLN" header elements being read from the SQLDA:

**DB2**

```

00100 TEXT SQL_TEXT(254)
00110 TEXT EMPL_NAME(30),EMPL_STREET(30),EMPL_STATE(2)
00120 BIG EMPL_ZIP_CODE
00130 SQL_TEXT="SELECT NAME, STREET, STATE, ZIPCODE"
00140 SQL_TEXT=SQL_TEXT+" FROM EMPLOYEE.TABLE"
00150 SQLDA("SQLDA1")=NEW
00160 IF SQLDA("SQLDA1","SQLN")<4
00170 .SQLDA("SQLDA1","SQLN")=4
00180 END
00190 SQLDA("SQLDA1","SQLD")=4
00200 EXEC_SQL:| PREPARE S1 FROM: SQL_TEXT
00210 END
00220 EXEC_SQL:| DESCRIBE S1 INTO SQLDA1
00230 END
00240 EXEC_SQL:| DECLARE C1 CURSOR FOR S1
00250 END
00260 EXEC_SQL:| OPEN C1
00270 END
00280 EXEC_SQL:| FETCH C1 USING DESCRIPTOR SQLDA1
00290 END
00300 EMPL_NAME=SQLDA("SQLDA1","SQLDATA",1)
00310 EMPL_STREET=SQLDA("SQLDA1","SQLDATA",2)
00320 EMPL_STATE=SQLDA("SQLDA1","SQLDATA",3)
00330 EMPL_ZIP_CODE=SQLDA("SQLDA1","SQLDATA",4)
00340 SQLDA("SQLDA1")=QUIT

```

**SUPRA**

```
00100 TEXT SQL_TEXT(254)
00110 TEXT EMPL_NAME(30),EMPL_STREET(30),EMPL_STATE(2)
00120 BIG EMPL_ZIP_CODE
00130 SQL_TEXT="SELECT NAME, STREET, STATE, ZIPCODE"
00140 SQL_TEXT=SQL_TEXT+" FROM EMPLOYEE.TABLE"
00150 SQLDA("SQLDA1")=NEW
00160 IF SQLDA("SQLDA1","SQLMAX")<4
00170 .SQLDA("SQLDA1","SQLMAX")=4
00180 END
00190 SQLDA("SQLDA1","SQLN")=4
00200 EXEC_SQL:| PREPARE S1 FROM: SQL_TEXT
00210 END
00220 EXEC_SQL:| DECLARE C1 CURSOR FOR S1
00230 END
00240 EXEC_SQL:| OPEN C1
00250 END
00260 EXEC_SQL:| PREPARE S2 FROM 'FETCH C1 USING DESCRIPTOR SQLDA1'
00270 END
00280 EXEC_SQL:| DESCRIBE S2 INTO SQLDA1
00290 END
00300 EXEC_SQL:| EXECUTE S2 USING DESCRIPTOR SQLDA1
00310 END
00320 EMPL_NAME=SQLDA("SQLDA1","SQLHOSTVAR",1)
00330 EMPL_STREET=SQLDA("SQLDA1","SQLHOSTVAR",2)
00340 EMPL_STATE=SQLDA("SQLDA1","SQLHOSTVAR",3)
00350 EMPL_ZIP_CODE=SQLDA("SQLDA1","SQLHOSTVAR",4)
00360 SQLDA("SQLDA1")=QUIT
```

---

## Read repeating elements

---

**SQLDA(*sqlda\_name*, *sqlda\_repeating\_element*, *index*)**

---

---

### *sqlda\_name*

- Description**     *Required.* Specifies the name of the SQLDA to be accessed.
- Format**             A text expression of 1–18 characters
- Consideration**   Naming conventions for SQL entities must be followed. Refer to the appropriate SQL language manual for the SQL database in use.
- 

### *sqlda\_repeating\_element*

- Description**     *Required.* Specifies the repeating element of the SQLDA that is accessed.
- Format**             A text expression that evaluates to one of the SQLDA repeating element names. See the “DB2” table on page 442 or the “SUPRA” table on page 443.
- 

### *index*

- Description**     *Required* when accessing repeating elements. Specifies the group of SQLDA repeating elements that is accessed.
- Format**             A numeric expression that evaluates between one and the maximum number of repeating groups currently in the SQLDA (SQLN or SQLMAX), inclusive

**Examples**

In the following examples, lines 300–330 (DB2) and lines 320–350 (SUPRA) show SQLDA repeating elements being moved from the SQLDA into the MANTIS program:

**DB2**

```
00100 TEXT SQL_TEXT(254)
00110 TEXT EMPL_NAME(30),EMPL_STREET(30),EMPL_STATE(2)
00120 BIG EMPL_ZIP_CODE
00130 SQL_TEXT="SELECT NAME, STREET, STATE, ZIPCODE"
00140 SQL_TEXT=SQL_TEXT+" FROM EMPLOYEE.TABLE"
00150 SQLDA("SQLDA1")=NEW
00160 IF SQLDA("SQLDA1","SQLN")<4
00170 .SQLDA("SQLDA1","SQLN")=4
00180 END
00190 SQLDA("SQLDA1","SQLD")=4
00200 EXEC_SQL:| PREPARE S1 FROM: SQL_TEXT
00210 END
00220 EXEC_SQL:| DESCRIBE S1 INTO SQLDA1
00230 END
00240 EXEC_SQL:| DECLARE C1 CURSOR FOR S1
00250 END
00260 EXEC_SQL:| OPEN C1
00270 END
00280 EXEC_SQL:| FETCH C1 USING DESCRIPTOR SQLDA1
00290 END
00300 EMPL_NAME=SQLDA("SQLDA1","SQLDATA",1)
00310 EMPL_STREET=SQLDA("SQLDA1","SQLDATA",2)
00320 EMPL_STATE=SQLDA("SQLDA1","SQLDATA",3)
00330 EMPL_ZIP_CODE=SQLDA("SQLDA1","SQLDATA",4)
00340 SQLDA("SQLDA1")=QUIT
```

**SUPRA**

```
00100 TEXT SQL_TEXT(254)
00110 TEXT EMPL_NAME(30),EMPL_STREET(30),EMPL_STATE(2)
00120 BIG EMPL_ZIP_CODE
00130 SQL_TEXT="SELECT NAME, STREET, STATE, ZIPCODE"
00140 SQL_TEXT=SQL_TEXT+" FROM EMPLOYEE.TABLE"
00150 SQLDA("SQLDA1")=NEW
00160 IF SQLDA("SQLDA1","SQLMAX")<4
00170 .SQLDA("SQLDA1","SQLMAX")=4
00180 END
00190 SQLDA("SQLDA1","SQLN")=4
00200 EXEC_SQL:| PREPARE S1 FROM: SQL_TEXT
00210 END
00220 EXEC_SQL:| DECLARE C1 CURSOR FOR S1
00230 END
00240 EXEC_SQL:| OPEN C1
00250 END
00260 EXEC_SQL:| PREPARE S2 FROM 'FETCH C1 USING DESCRIPTOR
SQLDA1'
00270 END
00280 EXEC_SQL:| DESCRIBE S2 INTO SQLDA1
00290 END
00300 EXEC_SQL:| EXECUTE S2 USING DESCRIPTOR SQLDA1
00310 END
00320 EMPL_NAME=SQLDA("SQLDA1","SQLHOSTVAR",1)
00330 EMPL_STREET=SQLDA("SQLDA1","SQLHOSTVAR",2)
00340 EMPL_STATE=SQLDA("SQLDA1","SQLHOSTVAR",3)
00350 EMPL_ZIP_CODE=SQLDA("SQLDA1","SQLHOSTVAR",4)
00360 SQLDA("SQLDA1")=QUIT
```

---

# SQLDA (Statement)

SQLDA is both a statement and a function. The SQLDA statement stores data from the MANTIS program into the SQL Descriptor Area (SQLDA). The SQLDA function transfers data from the SQLDA into the MANTIS program.

The SQLDA statements are shown below. The SQLDA statement is used to allocate or deallocate an SQLDA, and to transfer data from a MANTIS program into an SQLDA.

## Allocate an SQLDA

---

**SQLDA(*sqlda\_name*) = NEW**

---

---

### *sqlda\_name*

- Description**     *Required.* Specifies the name of the SQLDA to be allocated.
- Format**             A text expression of 1–18 characters
- Consideration**     Naming conventions for SQL entities must be followed. Refer to the appropriate SQL language manual for the SQL database in use.



**Examples**

The following examples for DB2 and for SUPRA show how to allocate an SQLDA. Line 150 in both examples allocates an SQLDA named the "SQLDA1".

**DB2**

```

00100 TEXT SQL_TEXT(254)
00110 TEXT EMPL_NAME(30),EMPL_STREET(30),EMPL_STATE(2)
00120 BIG EMPL_ZIP_CODE
00130 SQL_TEXT="SELECT NAME, STREET, STATE, ZIPCODE"
00140 SQL_TEXT=SQL_TEXT+" FROM EMPLOYEE.TABLE"
00150 SQLDA("SQLDA1")=NEW
00160 IF SQLDA("SQLDA1","SQLN")<4
00170 .SQLDA("SQLDA1","SQLN")=4
00180 END
00190 SQLDA("SQLDA1","SQLD")=4
00200 EXEC_SQL:| PREPARE S1 FROM: SQL_TEXT
00210 END
00220 EXEC_SQL:| DESCRIBE S1 INTO SQLDA1
00230 END
00240 EXEC_SQL:| DECLARE C1 CURSOR FOR S1
00250 END
00260 EXEC_SQL:| OPEN C1
00270 END
00280 EXEC_SQL:| FETCH C1 USING DESCRIPTOR SQLDA1
00290 END
00300 EMPL_NAME=SQLDA("SQLDA1","SQLDATA",1)
00310 EMPL_STREET=SQLDA("SQLDA1","SQLDATA",2)
00320 EMPL_STATE=SQLDA("SQLDA1","SQLDATA",3)
00330 EMPL_ZIP_CODE=SQLDA("SQLDA1","SQLDATA",4)
00340 SQLDA("SQLDA1")=QUIT

```

**SUPRA**

```
00100 TEXT SQL_TEXT(254)
00110 TEXT EMPL_NAME(30),EMPL_STREET(30),EMPL_STATE(2)
00120 BIG EMPL_ZIP_CODE
00130 SQL_TEXT="SELECT NAME, STREET, STATE, ZIPCODE"
00140 SQL_TEXT=SQL_TEXT+" FROM EMPLOYEE.TABLE"
00150 SQLDA("SQLDA1")=NEW
00160 IF SQLDA("SQLDA1","SQLMAX")<4
00170 .SQLDA("SQLDA1","SQLMAX")=4
00180 END
00190 SQLDA("SQLDA1","SQLN")=4
00200 EXEC_SQL:| PREPARE S1 FROM: SQL_TEXT
00210 END
00220 EXEC_SQL:| DECLARE C1 CURSOR FOR S1
00230 END
00240 EXEC_SQL:| OPEN C1
00250 END
00260 EXEC_SQL:| PREPARE S2 FROM 'FETCH C1 USING DESCRIPTOR SQLDA1'
00270 END
00280 EXEC_SQL:| DESCRIBE S2 INTO SQLDA1
00290 END
00300 EXEC_SQL:| EXECUTE S2 USING DESCRIPTOR SQLDA1
00310 END
00320 EMPL_NAME=SQLDA("SQLDA1","SQLHOSTVAR",1)
00330 EMPL_STREET=SQLDA("SQLDA1","SQLHOSTVAR",2)
00340 EMPL_STATE=SQLDA("SQLDA1","SQLHOSTVAR",3)
00350 EMPL_ZIP_CODE=SQLDA("SQLDA1","SQLHOSTVAR",4)
00360 SQLDA("SQLDA1")=QUIT
```

---

## Deallocate an SQLDA

---

**SQLDA(*sqlda\_name*) = QUIT**

---

---

***sqlda\_name***

**Description**     *Required.* Specifies the name of the SQLDA to be deallocated.

**Format**            A text expression of 1–18 characters

**Considerations**

- ◆ Naming conventions for SQL entities must be followed. Refer to the appropriate SQL language manual for the SQL database in use.
- ◆ Must be a previously defined SQLDA via SQLDA(*sqlda\_name*)=NEW.

**Examples**

The following examples show how to deallocate an SQLDA. Line 340 (DB2) and line 360 (SUPRA) deallocate an SQLDA named "SQLDA1":

**DB2**

```
00100 TEXT SQL_TEXT(254)
00110 TEXT EMPL_NAME(30),EMPL_STREET(30),EMPL_STATE(2)
00120 BIG EMPL_ZIP_CODE
00130 SQL_TEXT="SELECT NAME, STREET, STATE, ZIPCODE"
00140 SQL_TEXT=SQL_TEXT+" FROM EMPLOYEE.TABLE"
00150 SQLDA("SQLDA1")=NEW
00160 IF SQLDA("SQLDA1","SQLN")<4
00170 .SQLDA("SQLDA1","SQLN")=4
00180 END
00190 SQLDA("SQLDA1","SQLD")=4
00200 EXEC_SQL:| PREPARE S1 FROM: SQL_TEXT
00210 END
00220 EXEC_SQL:| DESCRIBE S1 INTO SQLDA1
00230 END
00240 EXEC_SQL:| DECLARE C1 CURSOR FOR S1
00250 END
00260 EXEC_SQL:| OPEN C1
00270 END
00280 EXEC_SQL:| FETCH C1 USING DESCRIPTOR SQLDA1
00290 END
00300 EMPL_NAME=SQLDA("SQLDA1","SQLDATA",1)
00310 EMPL_STREET=SQLDA("SQLDA1","SQLDATA",2)
00320 EMPL_STATE=SQLDA("SQLDA1","SQLDATA",3)
00330 EMPL_ZIP_CODE=SQLDA("SQLDA1","SQLDATA",4)
00340 SQLDA("SQLDA1")=QUIT
```

**SUPRA**

```
00100 TEXT SQL_TEXT(254)
00110 TEXT EMPL_NAME(30),EMPL_STREET(30),EMPL_STATE(2)
00120 BIG EMPL_ZIP_CODE
00130 SQL_TEXT="SELECT NAME, STREET, STATE, ZIPCODE"
00140 SQL_TEXT=SQL_TEXT+" FROM EMPLOYEE.TABLE"
00150 SQLDA("SQLDA1")=NEW
00160 IF SQLDA("SQLDA1","SQLMAX")<4
00170 .SQLDA("SQLDA1","SQLMAX")=4
00180 END
00190 SQLDA("SQLDA1","SQLN")=4
00200 EXEC_SQL:| PREPARE S1 FROM: SQL_TEXT
00210 END
00220 EXEC_SQL:| DECLARE C1 CURSOR FOR S1
00230 END
00240 EXEC_SQL:| OPEN C1
00250 END
00260 EXEC_SQL:| PREPARE S2 FROM 'FETCH C1 USING DESCRIPTOR SQLDA1'
00270 END
00280 EXEC_SQL:| DESCRIBE S2 INTO SQLDA1
00290 END
00300 EXEC_SQL:| EXECUTE S2 USING DESCRIPTOR SQLDA1
00310 END
00320 EMPL_NAME=SQLDA("SQLDA1","SQLHOSTVAR",1)
00330 EMPL_STREET=SQLDA("SQLDA1","SQLHOSTVAR",2)
00340 EMPL_STATE=SQLDA("SQLDA1","SQLHOSTVAR",3)
00350 EMPL_ZIP_CODE=SQLDA("SQLDA1","SQLHOSTVAR",4)
00360 SQLDA("SQLDA1")=QUIT
```

## Set header information

**SQLDA(*sqlda\_name*, *sqlda\_header\_element*)= *expression***

---

### *sqlda\_name*

- Description**     *Required.* Specifies the name of the SQLDA to be accessed.
- Format**             A text expression of 1–18 characters
- Considerations**
- ◆ Naming conventions for SQL entities must be followed. Refer to the appropriate SQL language manual for the SQL database in use.
  - ◆ Must be a previously defined SQLDA via SQLDA(*sqlda\_name*)=NEW.

---

### *sqlda\_header\_element*

- Description**     *Required.* Specifies the SQLDA header element that is accessed.
- Format**             A text expression that evaluates to one of the SQLDA header element names shown in the [table](#) on page 442 or the [table](#) on page 443 (SUPRA)

---

**expression**

**Description**     *Required.* Specifies the data to be transferred from the MANTIS program into the SQLDA.

**Format**            Must be consistent with the datatype of the SQLDA element being stored (either text or numeric)

**Considerations**

- ◆ Certain SQLDA elements are read-only and cannot have data from the MANTIS program stored in them. In some cases, storing data in one SQLDA element causes MANTIS to automatically update other SQLDA elements. For additional information, refer to *MANTIS DB2 Programming, OS/390, VSE/ESA*, P39-5028 or *MANTIS SUPRA SQL Programming, OS/390, VSE/ESA*, P39-3105.
- ◆ Some SQLDA element names are common to all SQL databases supported by MANTIS. However, the data contained in these elements is not always the same. For example, “SQLN” is the SQLDA element specifying the number of repeating groups contained in the SQLDA for DB2. “SQLMAX” is the equivalent SQLDA element for SUPRA. “SQLN” is an element in a SUPRA SQLDA, but it contains the number of repeating groups which are *in use*, not the maximum number of groups available. “SQLN” in the SUPRA SQLDA is equivalent to “SQLD” in the DB2 SQLDA.

**Examples**

In the following examples, for DB2 and for SUPRA, SQLDA header elements are set in lines 250 through 270:

**DB2**

```
00100 TEXT SQL_TEXT(254)
00110 TEXT EMPL_NAME(30),EMPL_STREET(30),EMPL_STATE(2)
00120 BIG EMPL_ZIP_CODE
00130 EMPL_NAME="JONES"
00140 EMPL_STREET=" NEW STREET ADDRESS"
00150 EMPL_STATE="OH"
00160 EMPL_ZIP_CODE="12345"
00170 SQL_TEXT="UPDATE EMPLOYEE.TABLE SET"
00180 SQL_TEXT=SQL_TEXT+" NAME = ?, STREET = ?, "
00190 SQL_TEXT=SQL_TEXT+" STATE = ?, ZIP_CODE = ?"
00200 SQL_TEXT=SQL_TEXT+" WHERE NAME = ?"
00210 EXEC_SQL:| PREPARE S1 FROM :SQL_TEXT
00220 END
00230 SQLDA("SQLDA1")=NEW
00240 IF SQLDA("SQLDA1","SQLN")<5
00250 .SQLDA("SQLDA1","SQLN")=5
00260 END
00270 SQLDA("SQLDA1","SQLD")=5
00280 SQLDA("SQLDA1","SQLDATA",1)=EMPL_NAME
00290 SQLDA("SQLDA1","SQLDATA",2)=EMPL_STREET
00300 SQLDA("SQLDA1","SQLDATA",3)=EMPL_STATE
00310 SQLDA("SQLDA1","SQLDATA",4)=EMPL_ZIP_CODE
00320 SQLDA("SQLDA1","SQLDATA",5)=EMPL_NAME
00330 EXEC_SQL:| EXECUTE S1 USING DESCRIPTOR SQLDA1
00340 END
00350 SQLDA("SQLDA1")=QUIT
```



**SUPRA**

```
00100 TEXT SQL_TEXT(254)
00110 TEXT EMPL_NAME(30),EMPL_STREET(30),EMPL_STATE(2)
00120 BIG EMPL_ZIP_CODE
00130 EMPL_NAME="JONES"
00140 EMPL_STREET=" NEW STREET ADDRESS"
00150 EMPL_STATE="OH"
00160 EMPL_ZIP_CODE="12345"
00170 SQL_TEXT="UPDATE EMPLOYEE.TABLE SET"
00180 SQL_TEXT=SQL_TEXT+" NAME = ?, STREET = ?, "
00190 SQL_TEXT=SQL_TEXT+" STATE = ?, ZIP_CODE = ?"
00200 SQL_TEXT=SQL_TEXT+" WHERE NAME = ?"
00210 EXEC_SQL:| PREPARE S1 FROM :SQL_TEXT
00220 END
00230 SQLDA ("SQLDA1")=NEW
00240 IF SQLDA("SQLDA1","SQLMAX")<5
00250 .SQLDA("SQLDA1","SQLMAX")=5
00260 END
00270 SQLDA("SQLDA1","SQLN")=5
00280 SQLDA("SQLDA1","SQLHOSTVAR",1)=EMPL_NAME
00290 SQLDA("SQLDA1","SQLHOSTVAR",2)=EMPL_STREET
00300 SQLDA("SQLDA1","SQLHOSTVAR",3)=EMPL_STATE
00310 SQLDA("SQLDA1","SQLHOSTVAR",4)=EMPL_ZIP_CODE
00320 SQLDA("SQLDA1","SQLHOSTVAR",5)=EMPL_NAME
00330 EXEC_SQL:| EXECUTE S1 USING DESCRIPTOR SQLDA1
00340 END
00350 SQLDA("SQLDA1")=QUIT
```

The following table lists and describes the SQLDA header elements for DB2:

Element name	MANTIS compatible datatype	Contents / considerations	Updateable?
SQLDAID	TEXT(8)	Eyecatcher. Set by SQL.	No
SQLDABC	BIG	SQLDA length. Set by SQL when the SQLDA is allocated; modified when SQLN is changed.	Yes
SQLN	BIG	Number of repeating groups (SQLVAR) in the SQLDA. Set using installation defined default value when the SQLDA is allocated. Can be modified by the MANTIS program if needed.	Yes
SQLD	BIG	Number of repeating groups currently in use. Set by SQL as the result of a DESCRIBE; can be set by program when necessary.	Yes

The following table lists and describes the SQLDA header elements for SUPRA:

Element name	MANTIS compatible datatype	Contents / considerations	Updateable?
SQLDAID	TEXT(8)	Eyecatcher. Set by SQL.	No
SQLABC	BIG	Length of SQLCA. Set by SQL.	No
SQLMAX	BIG	Number of repeating groups (SQLVAR) in the SQLDA. Set using installation defined default value when the SQLDA is allocated. Can be modified by the MANTIS program if needed.	Yes
SQLN	BIG	Total number of repeating groups in use in the SQLDA. Set as a result of a DESCRIBE to the total number of host variable parameters in the statement (except for DESCRIBE in FETCH USING DESCRIPTOR where SQLN is set to the number of result table columns).	Yes
SQLD	BIG	Total number of output host variables in the SQLDA. Set as a result of a DESCRIBE to the number of host variables (except for DESCRIBE in FETCH USING DESCRIPTOR where SQLD is set to the number of result table columns).	Yes

## Set repeating element information

**SQLDA(*sqlda\_name*, *repeating\_element*, *index*)= *expression***

---

### *sqlda\_name*

- Description**     *Required.* Specifies the name of the SQLDA to be accessed.
- Format**            A text expression of 1–18 characters
- Consideration**   Naming conventions for SQL entities must be followed. Refer to the appropriate SQL language manual for the SQL database in use.

---

### *sqlda\_repeating\_element*

- Description**     *Required.* Specifies the repeating element of the SQLDA that is accessed.
- Format**            A text expression that evaluates to one of the SQLDA repeating element names shown in the following tables
- Consideration**   Repeating element names must be selected from the list for the SQL database in use on your system. See the first table at the end of this section (DB2) or the second table at the end of this section (SUPRA).

---

**index**

- Description**     *Required* when accessing repeating elements. Specifies the group of SQLDA repeating elements that is accessed.
- Format**            A numeric expression between one and the maximum number of repeating groups currently in the SQLDA (SQLN or SQLMAX), inclusive

**General considerations**

- ◆ Using certain SQLDA elements causes other SQLDA elements to automatically be set by MANTIS. For example, storing data into the SQLDA element “SQLDATA” causes MANTIS to set the datatype (“SQLTYPE”) and data length (“SQLLEN”) elements automatically. Storing data into the SQLDA “SQLIND” element updates the datatype element (“SQLTYPE”) to show that an indicator variable is present. For more information, and for considerations when using the SQLDA function, refer to *MANTIS DB2 Programming, OS/390, VSE/ESA*, P39-5028, or *MANTIS SUPRA SQL Programming, OS/390, VSE/ESA*, P39-3105.
- ◆ See also “EXEC\_SQL-END” on page 217, “SQLCA (Function)” on page 420, and “SQLCA (Statement)” on page 424.

**Examples**

In the following examples for DB2 and for SUPRA, lines 280–320 store data from the MANTIS program into SQLDA repeating group elements:

**DB2**

```
00100 TEXT SQL_TEXT(254)
00110 TEXT EMPL_NAME(30),EMPL_STREET(30),EMPL_STATE(2)
00120 BIG EMPL_ZIP_CODE
00130 EMPL_NAME="JONES"
00140 EMPL_STREET=" NEW STREET ADDRESS"
00150 EMPL_STATE="OH"
00160 EMPL_ZIP_CODE="12345"
00170 SQL_TEXT="UPDATE EMPLOYEE.TABLE SET"
00180 SQL_TEXT=SQL_TEXT+" NAME = ?, STREET = ?, "
00190 SQL_TEXT=SQL_TEXT+" STATE = ?, ZIP_CODE = ?"
00200 SQL_TEXT=SQL_TEXT+" WHERE NAME = ?"
00210 EXEC_SQL:| PREPARE S1 FROM :SQL_TEXT
00220 END
00230 SQLDA("SQLDA1")=NEW
00240 IF SQLDA("SQLDA1","SQLN")<5
00250 .SQLDA("SQLDA1","SQLN")=5
00260 END
00270 SQLDA("SQLDA1","SQLD")=5
00280 SQLDA("SQLDA1","SQLDATA",1)=EMPL_NAME
00290 SQLDA("SQLDA1","SQLDATA",2)=EMPL_STREET
00300 SQLDA("SQLDA1","SQLDATA",3)=EMPL_STATE
00310 SQLDA("SQLDA1","SQLDATA",4)=EMPL_ZIP_CODE
00320 SQLDA("SQLDA1","SQLDATA",5)=EMPL_NAME
00330 EXEC_SQL:| EXECUTE S1 USING DESCRIPTOR SQLDA1
00340 END
00350 SQLDA("SQLDA1")=QUIT
```

**SUPRA**

```

00100 TEXT SQL_TEXT(254)
00110 TEXT EMPL_NAME(30),EMPL_STREET(30),EMPL_STATE(2)
00120 BIG EMPL_ZIP_CODE
00130 EMPL_NAME="JONES"
00140 EMPL_STREET=" NEW STREET ADDRESS"
00150 EMPL_STATE="OH"
00160 EMPL_ZIP_CODE="12345"
00170 SQL_TEXT="UPDATE EMPLOYEE.TABLE SET"
00180 SQL_TEXT=SQL_TEXT+" NAME = ?, STREET = ?,"
00190 SQL_TEXT=SQL_TEXT+" STATE = ?, ZIP_CODE = ?"
00200 SQL_TEXT=SQL_TEXT+" WHERE NAME = ?"
00210 EXEC_SQL:| PREPARE S1 FROM :SQL_TEXT
00220 END
00230 SQLDA ("SQLDA1")=NEW
00240 IF SQLDA("SQLDA1","SQLMAX")<5
00250 .SQLDA("SQLDA1","SQLMAX")=5
00260 END
00270 SQLDA("SQLDA1","SQLN")=5
00280 SQLDA("SQLDA1","SQLHOSTVAR",1)=EMPL_NAME
00290 SQLDA("SQLDA1","SQLHOSTVAR",2)=EMPL_STREET
00300 SQLDA("SQLDA1","SQLHOSTVAR",3)=EMPL_STATE
00310 SQLDA("SQLDA1","SQLHOSTVAR",4)=EMPL_ZIP_CODE
00320 SQLDA("SQLDA1","SQLHOSTVAR",5)=EMPL_NAME
00330 EXEC_SQL:| EXECUTE S1 USING DESCRIPTOR SQLDA1
00340 END
00350 SQLDA("SQLDA1")=QUIT

```

The following table lists the SQLDA repeating elements in DB2:

Element name	MANTIS compatible datatype	Contents / considerations	Updateable?
SQLTYPE	BIG	SQL datatype code. Code differs depending on whether set by SQLDATA or SQLIND. Set when the value of variable is transferred via SQLDATA or SQLIND.	No
SQLEN	BIG	Length of data element in current repeating group. Set when value of variable is transferred via SQLDATA.	No
SQLDATA	TEXT, BIG,	Subfunction that moves or sets DBCS data, sets SQLTYPE and SQLEN, and sets address of the data in the SQLDA. Used to transfer value of variable between database and MANTIS.	Yes
SQLIND	BIG	Subfunction that moves data, sets SQLTYPE and address of the data in the SQLDA. Used to transfer value of the indicator variable between database and MANTIS.	Yes
SQLNAME	TEXT(30)	SQL column name. Set by SQL, can be reset by the MANTIS program.	Yes



Element name	MANTIS compatible datatype	Contents / considerations	Updateable?
SQLCOLNAME	TEXT(18)	SQL column name. Set by SUPRA as the result of a DESCRIBE; can be set by the MANTIS program.	Yes
SQLCOLIO	BIG	Indicates whether host variable is input or output. Set by SUPRA as the result of a DESCRIBE	No
SQLCOLMODE	BIG	Indicates whether null values are allowed. Set as a result of a DESCRIBE.	No
SQLCOLTYPE	BIG	Datatype as it resides on the database. Set by SUPRA as a result of a DESCRIBE.	No
SQLCOLLENGTH	BIG	Total number of bytes used to store the data. Set by SUPRA as the result of a DESCRIBE, or by MANTIS when data is transferred by SQLHOSTVAR.	No
SQLCOLFRAC	BIG	Number of decimal positions for FIXED column types. Set by SUPRA as the result of a DESCRIBE. Not used by MANTIS because all numeric data is floating point.	No

The following table lists the SQLDA repeating elements in SUPRA:

Element name	MANTIS compatible datatype	Contents / considerations	Updateable?
SQLHOSTIND	BIG	Contains the value of the indicator variable. Used to indicate the presence of null variables and truncated data. Set by SUPRA during SQL function; can be set by the MANTIS program.	Yes
SQLHOSTVARTY	BIG	Contains the datatype of the data in the SQLDA. Set by MANTIS when SQLHOSTVAR is used.	No
SQLHOSTVAR	TEXT, BIG, or DBCS	Subfunction that physically transfers data to MANTIS data areas and the between SQLDA data areas. Used to transfer value of variable between database and MANTIS.	Yes

# SQR

The SQR function returns the square root of an arithmetic expression.

**SQR(a)**

**a**

**Description**     *Required.* Specifies any valid non-negative arithmetic expression.

**General consideration**

See also “ABS” on page 86, “ATN” on page 94, “COS” on page 164, “INT” on page 294, “LOG” on page 320, “SGN” on page 399, “SIN” on page 403, “SQR” on page 451, and “TAN” on page 454.

**Example**     The following examples show how the SQR function returns the square root of an arithmetic expression:

Example	Results	Comments
SQR ( 25 )	5	
SQR ( 2 )	1.41421356	
SQR ( 0 )	0	

## STOP

The STOP statement terminates program execution. When a STOP statement is executed, MANTIS:

- ◆ Returns to programming mode if the program was executing while in programming mode.
- ◆ Returns to your Facility Selection menu if the program was executing while *not* in programming mode.

---

### STOP

---

#### General considerations

- ◆ STOP statements are used for step processing and debugging. When placed at strategic locations in a program, you can use STOP with a RUN statement and a statement number to execute and check a portion of a program. You can also use STOP to check variables, and then execute the next portion.
- ◆ In programming mode, a STOP in an external routine stops execution within the external routine. Use the immediate EXIT command to return to the calling program (an immediate-mode EXIT at DOLEVEL zero causes errors).
- ◆ STOP statements are required before the first internal subroutine ENTRY when the top-level routine is not surrounded by an ENTRY-EXIT pair.
- ◆ Use STOP to return to the facility program instead of reaching the top-level EXIT of a DOLEVEL 0 program.
- ◆ See also “EXIT” on page 219, “RETURN” on page 388, and “RUN” on page 391.

**Example**

The following example shows how the STOP statement is used to stop program execution:

```
00020 SCREEN MAP1( "INDEX" )
00030 FILE REC1( "INDEX" , "SERENDIPITY" )
00040 CONVERSE MAP1
00050 WHILE MAP1<>"CANCEL"
00060 .DO INSERT_RECORD
00065 .STOP
00070 .CLEAR MAP1
00080 .CONVERSE MAP1
00090 END
00100 STOP
00120 ENTRY INSERT_RECORD
00130 .INSERT REC1
00140 EXIT
```

*Check variables and return codes  
then RUN 70*

# TAN

The TAN function returns the tangent of *a* where *a* is in radians.

**TAN(*a*)**

*a*

**Description**     *Required.* Specifies the angle whose tangent is to be returned.

**Format**            Any valid arithmetic expression

**General considerations**

- ◆ The variable (*a*) must be a value where a tangent is defined and TAN(*a*) is within the numeric limits of MANTIS.
- ◆ If the angle is in degrees, it must be converted to radians. See “PI” on page 367 for instructions on how to do this.
- ◆ See also “ATN” on page 94, “COS” on page 164, “PI” on page 367, and “SIN” on page 403.

**Example**            The following example shows how the TAN function returns the tangent of an arithmetic expression:

Example	Results	Comments
TAN(100)	-.587213915	
TAN(0)	0	
TAN(PI/4)	1	

# TERMINAL

The `TERMINAL` function returns a text string of 1–8 characters containing the terminal ID.

## TERMINAL

### General considerations

- ◆ The value returned by `TERMINAL` varies depending on the operating environment. See the examples.
- ◆ See also “`PASSWORD`” on page 349, “`PRINTER (Function)`” on page 370, “`PRINTER (Statement)`” on page 371, “`TERMSIZE`” on page 456, and “`USER`” on page 497.

### Examples

The following show values returned by the `TERMINAL` function

Example	Results	Comments
<code>TERMINAL</code>	"L338"	Sample EIBTRMID for CICS.
<code>TERMINAL</code>	"BACK\$MAN"	Background task.
<code>TERMINAL</code>	"DUMMY"	Batch MANTIS.
<code>TERMINAL(1,1)</code>	"L"	Substringing OK.

The following example shows how the `TERMINAL` function can be used to test for a specific terminal ID and return a message:

```
00020 IF TERMINAL="XX02"
00030 .SHOW TERMINAL
00040 .SHOW"NOT AUTHORIZED FROM THIS TERMINAL":WAIT
00050 END
```

# TERMSIZE

The TERMSIZE function returns the size of the current terminal in rows and columns.

## TERMSIZE

### General considerations

- ◆ When TERMSIZE=*expr* is executed, a valid execution is accepted but ignored for compatibility with older releases of MANTIS.
- ◆ TERMSIZE is available for compatibility with previous releases of MANTIS. The ATTRIBUTE(TERMINAL) function can be used to obtain terminal row and column values as well as other terminal attributes.
- ◆ See also “ATTRIBUTE TERMINAL/CURSOR statement” on page 109, “PASSWORD” on page 349, “PRINTER (Function)” on page 370, “PRINTER (Statement)” on page 371, and “USER” on page 497.

### Examples

Example	Results	Comments
TERMSIZE	" 24X80 "	For a "model 2".
TERMSIZE ( 1 , 2 )	" 24 "	Number of rows.
TERMSIZE ( 4 )	" 80 "	Number of columns.



# TEXT

The TEXT statement names and specifies dimensions for text variables and lists.

---


$$\text{TEXT } name1 \left[ \left( \begin{array}{c} [n,] length \\ \underline{16} \end{array} \right) \right]$$

$$\left[ , name2 \left[ \left( \begin{array}{c} [n,] length \\ \underline{16} \end{array} \right) \right] \dots \right]$$


---

## *name*

- Description**    *Required.* Specifies the name of the text variable.
- Format**        A MANTIS symbolic name, (see “[Symbolic names](#)” on page 24)
- Consideration** When the symbolic name is previously defined, MANTIS bypasses this definition.

## *n*

- Description**    *Optional.* Specifies the number of elements in a text array.
- Format**        Arithmetic expression that evaluates to a value in the range 1–255
- Considerations**
- ◆ MANTIS rounds *n* to an integer value.
  - ◆ If not specified, *name* is a text scalar.

## *length*

- Description**    *Optional.* Specifies the maximum length of each text element.
- Format**        Arithmetic expression that evaluates to a value in the range 1–254
- Default**        16
- Consideration** MANTIS rounds *length* to an integer value.

## General considerations

- ◆ A TEXT variable contains a zero-length string (NULL), upon initial definition.
- ◆ MANTIS truncates trailing blanks for text fields when reading in from external files, and MANTIS adds trailing blanks when writing to external files. MANTIS Internal Files maintain trailing blanks and current length.
- ◆ Hexadecimal data can be stored/handled but might not display as intelligible data in a MANTIS text field. Each terminal control characters is translated to a question mark (“?”) for display.
- ◆ MANTIS accepts only as many characters in a text variable as you specify in the TEXT statement. See the following examples and “LET (TEXT/KANJI/DBCS variables)” on page 312.
- ◆ Text variables can be used to hold upper or lowercase characters.
- ◆ Text variables can be used to hold double-byte character set (DBCS) characters when used with Shift-out and Shift-in characters and MIXMODE ON.
- ◆ See also “BIG” on page 134, “DBCS considerations” on page 29, “DBCS (Statement)(Kanji users only)” on page 181 “KANJI (Kanji users only)” on page 298, “MIXMODE” on page 329, “POINT” on page 368, “SIZE” on page 404, and “SMALL” on page 415.

## Examples

- ◆ The following example shows how the TEXT retains only the length dimension specified. For example, if you enter:

```
00010 TEXT ALPHA      <== Defaults to length of 16
00020 TEXT BETA(5)
00030 ALPHA="123456789ABCDEFGHIJK"
00040 BETA="123456789"
00050 SHOW ALPHA
00060 SHOW BETA
00070 WAIT
```

the screen displays:

```
123456789ABCDEFG
12345
```

- ◆ The following example shows how the TEXT statement is used to define several different variables. In the following statement, ALPHA has a maximum length of 16:

```
00010 TEXT ALPHA
00020 TEXT BETA(20)
00030 TEXT GAMMA(10,30)
```

## TIME (Function)

TIME is both a statement and a function. The TIME function returns a text string containing the current time in the format of the current specification.

---

### TIME

---

#### General considerations

- ◆ The format of the date string can be an installation-defined value or a program-defined value (see “[TIME \(Statement\)](#)” on page 462). The supplied default is “HH:MM:SS”. The TIME statement and function provide flexibility in the choice of the delimiter that appears between the elements of the TIME text string. You may choose no delimiter or any delimiter, such as a colon or a hyphen. MANTIS will return the text string in whatever format you choose.
- ◆ The TIME function may have substring parameters (see “[Substringing text variables](#)” on page 53).
- ◆ See “[TIME \(Statement\)](#)” on page 462 for how to specify the format.

Examples

- ◆ The following examples show how the TIME function returns the current time:

Example	Results	Comments
TIME	"13:23:01"	Default format.
TIME	"1:23 PM"	Alternate format set up by: DATE="HH:MM PM"
TIME(1,5)	"13:23"	Substringing allowed; in this case, just the HH:MM.
TIME	"13:23"	When TIME="HH:MM".
TIME	"05:30:00"	When TIME="05:30:00", all digits are considered to be punctuation characters. Subsequent TIME functions return this value until the format reset.

# TIME (Statement)

TIME is both a statement and a function. Use the statement to specify a text string by which the TIME function formats the current time.

**TIME=mask-expression**

## mask-expression

**Description**     *Required.* Specifies the type of string MANTIS uses to return the current time with the TIME function.

**Format**            A text string of 0–12 characters

### Considerations

- ◆ If the text string specified is longer than 12 characters, the first 12 characters will be used.
- ◆ If NULL ("" ) is specified, the installation default is used.
- ◆ The following strings will be substituted with the data described when used with the TIME function:
  - AM =AM/PM indicator
  - HH = hours (12 hour/zero suppressed when AM is present, else 24-hour)
  - MM = minutes
  - SS = seconds
  - Punctuation characters



Your Master User may define an installation-wide default value for the TIME format.

## General considerations

- ◆ You can specify the format of the TIME returned value in one of the following ways:
  - TIME = *mask*, as specified in a program by a user.
  - TIME format specified by installation (see your Master User).
  - System Default (HH:MM:SS).
- ◆ The format is maintained down DO/CHAIN levels.
- ◆ No subscripting of the statement is permitted. (TIME(1,5)="HH:MM"—the TIME statement—is invalid.) However, you can use subscripts on the TIME function,
- ◆ When MANTIS executes a CHAIN (without LEVEL), KILL, fault (error), or STOP, the format is reset to the installation default.
- ◆ HH:MM:SS will be used for all CONTROL users programs.
- ◆ Arguments for the TIME statement are translated to uppercase upon execution of your program.
- ◆ During testing, a constant TIME can be set; for example, TIME="05:30:00". It is maintained according to the rules stated above.
- ◆ The TIME statement and function provide flexibility for choice of the delimiter that appears between the elements of the TIME text string. You may choose no delimiter, a slash, a hyphen, or a period. MANTIS will return the text string in whatever format you choose.
- ◆ See also "DATE (Function)" on page 177, "DATE (Statement)" on page 179, and "TIME (Function)" on page 460.

## Examples

See examples under "TIME (Function)" on page 460.

---

# TOTAL

## TOTAL (TOTAL and SUPRA PDM users only)

The TOTAL statement specifies a TOTAL file view.

```
TOTAL { name1 ([ library1: ] TOTAL - view1, password1[, PREFIX][, n1])  
      [ , name2 ([ library2: ] TOTAL - view2, password2[, PREFIX][, n2]) . . . ]  
      ON  
      OFF }
```

---

### *name*

- Description**     *Required.* Specifies the name you use to refer to a TOTAL file view in subsequent GET, UPDATE, INSERT, DELETE, and DEQUEUE statements.
- Format**            A MANTIS symbolic name, (see “[Symbolic names](#)” on page 24)
- Consideration**   When the symbolic name is previously defined, MANTIS bypasses this definition.



***[library:] TOTAL-view***

<b>Description</b>	<i>Required.</i> Specifies the name of an existing TOTAL file view as you defined it during TOTAL file view design.
<b>Format</b>	1–33 character text expression that evaluates to a valid TOTAL file view name

**Considerations**

- ◆ If the TOTAL view is in another user's library, you can access it by specifying the name of the user in whose library it does reside.  
*[library:] TOTAL-view*
- ◆ If the TOTAL view resides in your library, you can specify only the TOTAL view name.
- ◆ If you want this entity to be HPO bound, the library name is required, even if it is your own library.
- ◆ This parameter is translated to uppercase upon execution of your program.

---

***password***

<b>Description</b>	<i>Required.</i> Specifies the password valid for the type of access you need for this TOTAL file view (e.g., read-only, alter, insert/delete).
<b>Format</b>	Text expression that evaluates to the password; and can be entered in lower or uppercase

**PREFIX**

**Description**     *Optional.* Specifies whether MANTIS places the symbolic name and an underscore before all data field names associated with this TOTAL view, including the *refer variable*. If, for example, you code:

```
10 TOTAL CUSTOMER( "CLIENT" , "SALES" , PREFIX)
```

and the TOTAL view “CLIENT” had a data field named NUMBER, the program would refer to that data field now as CUSTOMER\_NUMBER.

**Format**             Must be coded exactly as shown

**Consideration** See the **PREFIX** considerations for “FILE” (“FILE” starts on page 222).

---

***n***

**Description**     *Optional.* Indicates how many buffers MANTIS should allocate. Each buffer contains data accessible through the TOTAL view from a single TOTAL record. MANTIS also allocates multiple buffers for the reference variable.

**Default**             1

**Format**             Arithmetic expression that evaluates to a value in the range 1–255

**Consideration** When you use the *n* parameter, any GET, INSERT, UPDATE, and DELETE statements should contain the LEVEL=*n* option to tell MANTIS which buffer to process.

---

**ON**

**Description**     *Optional.* Forces MANTIS to sign on to the TOTAL or SUPRA PDM database.

**Consideration** You may need to use this option if you are calling an INTERFACE program that expects a TOTAL SINON or does a SINOF before returning.

---

**OFF**

**Description**     *Optional.* Forces MANTIS to sign off from the TOTAL or SUPRA PDM database.

**Consideration** You may need to use this option if you are calling an INTERFACE program that expects to do a TOTAL SINON.

## General considerations

MANTIS retrieves the specified TOTAL file view from your library and validates it for consistency with the active Database Descriptor Module (DBMOD) or SUPRA directory. If there is an inconsistency, MANTIS returns an error message and halts execution.

- ◆ A TOTAL file open is issued on the first DELETE, GET, INSERT, or UPDATE.
- ◆ MANTIS opens only the necessary TOTAL database files with an access mode dependent on the password (for deletions, insertions, or updates—SUPD; for gets—IUPD).
- ◆ If the view applies to a single-entry TOTAL file, MANTIS opens only this file. The REALM mode is IUPD if you can issue only GET statements or SUPD if you can issue update commands.
- ◆ If the view applies to variable-entry TOTAL files, MANTIS opens with “IUPD” for read-only requests or with “SUPD” for updates. In addition, any associated master files are opened. If inserting or deleting is not allowed, MANTIS opens (in IUPD mode) only the master file necessary to provide keyed access. If inserting or deleting is allowed, MANTIS opens (in SUPD mode) all associated master files. The element list defined in the TOTAL file view is then bound using TOTAL’s binding conventions.
- ◆ In a DTB environment, you should COMMIT following TOTAL statements to avoid locking the system resource table.
- ◆ The *library* and *name* arguments for the TOTAL statement are translated into uppercase upon execution of your program.
- ◆ A TOTAL ON statement accepts a return status of “\*\*\*\*” and “ACTV”. Any other return status produces an error message.
- ◆ A TOTAL OFF statement accepts a return status of “\*\*\*\*”, “NCAT”, or “NOTO”. Any other return status produces an error message.
- ◆ See also “DELETE” on page 183, “GET” on page 234, “INSERT” on page 277, “TRAP” on page 469, and “UPDATE” on page 479.

## Examples

- ◆ The following example shows TOTAL statements to access two files:

```
00040 TOTAL CUSTOMER("CLIENT","SALES")
```

```
00050 TOTAL HISTORY("PAYMENTS","TEXAS",11)
```

If the SALES password allows viewing only, MANTIS opens the CUSTOMER file without TOTAL update access. HISTORY would be allocated 11 buffers. If the password allows updating, TOTAL update access is available.

- ◆ The following example shows how the TOTAL statement accesses a file:

```
00010 TOTAL CUSTOMER("CLIENT","SALES")
```

```
00020 TRAP CUSTOMER ON
```

```
00030 GET CUSTOMER
```

```
00040 IF CUSTOMER="NOTOPEN"
```

```
·
```

```
·
```

```
·
```

---

## TRAP

The TRAP statement intercepts I/O errors from MANTIS files, external VSAM files, or additional status codes returned by TOTAL or RDM. The program continues to execute for trapped errors.

---

**TRAP** *name* **ON**  
**OFF**

---

---

### *name*

**Description**     *Required.* Specifies the name of the file or view as defined in a FILE, ACCESS, TOTAL, or VIEW statement.

**Format**            A MANTIS symbolic name, (see “[Symbolic names](#)” on page 24)

---

### ON

### OFF

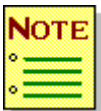
**Description**     *Required.* ON starts interception of errors; OFF halts the messages.

**Default**            OFF

**Format**            Must be coded exactly as shown

## General considerations

- ◆ With TRAP ON, MANTIS reflects the status of a GET, UPDATE, DELETE, and INSERT statement in the value of a symbolic variable *name*. See “GET” on page 234, “UPDATE” on page 479, “DELETE” on page 183, and “INSERT” on page 277.



---

Use caution when using TRAP because of the possible loss of database integrity if the appropriate action is not taken for trapped errors. Consult your Master User or DBA before you attempt to trap errors.

---

- ◆ TRAP ON and proper error-checking routines are recommended for all production programs.
- ◆ If you use the TRAP statement, you assume all responsibility for detecting and handling error situations that can occur while processing the file. Improper use can result in loop conditions. For example:

```
00010 ACCESS REC( "X", "X" )
00020 TRAP REC ON
00030 GET REC FIRST
00040 WHILE REC<>"END"
00050 .DELETE REC
00060 .GET REC
00070 END
```

This program can result in a loop if, for example, an “ERROR”, “LOCK”, or “NOTOPEN” status is obtained on a GET.

- ◆ Problems can occur in other situations where status is not checked:

```
00010 ACCESS RECA( "A","A"), RECB("B","B"): | A and B have
                                              fields in common
```

```
00020 TRAP RECA ON:TRAP RECB ON
```

```
00030 GET RECA
```

```
00040 INSERT RECB
```

If the GET for RECA fails, the resulting insert to RECB might not contain the desired results. Note that the insert on RECB can also fail without obtaining an error message.

- ◆ The status “NOTOPEN” is only issued on the first use after the ACCESS statement. On subsequent use, an “ERROR” status is issued if the file cannot be opened.
- ◆ See also “ACCESS” on page 87, “ASI” on page 93, “DELETE” on page 183, “FILE” on page 222, “FSI” on page 232, “GET” on page 234, “INSERT” on page 277, “RELEASE (Function)” on page 378, “RELEASE (Statement)” on page 380, “TOTAL” on page 464, “VIEW” on page 501, and “UPDATE” on page 479.

### Example

The following example shows how the TRAP statement allows status checking:

```
00100 FILE REC("EXAMPLES:CUSTOMERS","CASINO")
```

```
.
```

```
.
```

```
.
```

```
00160 TRAP REC ON
```

```
00170 INSERT REC
```

```
.
```

```
.
```

```
.
```

```
00220 UPDATE REC
```

```
00230 IF REC="ERROR"
```

```
00240 .RESET
```

```
00250 END
```

```
.
```

```
.
```

```
.
```

# TRUE

The TRUE function returns a value of +1 and can be used to set conditions within your program.

## TRUE

### General considerations

- ◆ Any nonzero value in a relational expression evaluates to “TRUE” condition. For example, you can code IF ERROR\_CONDITION and not IF ERROR\_CONDITION = TRUE. The latter will be false if ERROR\_CONDITION has any value except +1.
- ◆ See also “FALSE” on page 221, “MODIFIED” (explicit use of TRUE function) on page 332, “NOT” on page 336, and “NULL” on page 338.

### Example

Example	Results	Comments
TRUE	1	

The following example shows how the TRUE function can be used to set conditions in a program:

```
00010 ERROR_OCCURED=FALSE
.
.
.
00050 IF CUST_NUM>10000
00060 .ERROR OCCURED=TRUE
00070 END
.
.
.
00100 IF ERROR_OCCURED
00110 .DO ERROR_DISPLAY
00120 .ELSE
00130 .DO VALID_DISPLAY
00140 END
```



# TXT

The TXT function returns the text value of a numeric expression, a.

**TXT(a)**

**a**

**Description**     *Required.* Specifies any valid arithmetic expression.

**General considerations**

- ◆ TXT(0) returns a single space. SIZE (TXT(0)) is equal to 1. It does not return an empty string (NULL) or "0".
- ◆ When necessary, MANTIS will return the TXT value in E-notation.
- ◆ IBM MANTIS does not place a leading 0 before nonzero values between -1 and 1. MANTIS in a non-IBM environment does place a leading 0 before nonzero values between -1 and 1.
- ◆ See also "FORMAT" on page 230 and "VALUE" on page 499.

**Example**

Example	Results	Comments
TXT(123)	"123"	
TXT(+123)	"123"	
TXT(-456)	"-456"	
TXT(1/2)	".5"	Platform differences
TXT(-1/3)	"-.333333333"	Platform differences
TXT(EXP(30))	".106864746E14"	E-notation
TXT(0)	" "	Blank, not null
TXT(-EXP(-30))	"-.935762297E-13"	E-notation

The following example shows how the TEXT function can be used to create a correlation between PF keys and option numbers:

```
00010 KEY_TO_PRESS="PF"+TXT(OPTION_NUMBER)
```

## UNPAD

The UNPAD statement allows you to remove all occurrences of a specified character from either or both sides of a text or DBCS variable.

---

<b>UNPAD v [exp]</b>	<b>BEFORE</b>
	<b>AFTER</b>
	<b>ALL</b>

---

**v**

**Description**     *Required.* Represents the variable to be unpadded.

### Considerations

- ◆ If the referenced variable is subscripted (apart from the array occurrence) (see the LET statement), the BEFORE, AFTER, and ALL options cannot be used. If you try to use the substring subscripts with one of these options, you receive an error message.
- ◆ If the referenced variable has two substring subscripts, each subscript represents the boundaries of the unpad operation. The leftmost boundary is marked by the first substring subscript; the rightmost boundary by the second substring subscript.
- ◆ If the referenced variable has one substring subscript, MANTIS assumes that the second (missing) subscript is equal to the current size of the variable. Therefore, the first substring subscript marks the starting point of the unpad operation. With no second substring subscript, the end of the unpad operation is the rightmost byte of the originally defined area for the variable.
- ◆ MANTIS unpads the variable by comparing each character in the variable with the unpad character. If the two are equal, the matching character is stripped off and the rest of the text is moved to the left. The current length is reduced by 1. This process repeats until a character not equal to the unpad character is detected. Characters are checked from left to right if you use the BEFORE option; from right to left if you use the AFTER option; and a combination with the ALL option.

---

**exp**

<b>Description</b>	<i>Optional.</i> Indicates a text or DBCS expression that represents the character to be unpadded.
<b>Default</b>	Spaces (blanks)
<b>Format</b>	1– <i>n</i> character expression for text or DBCS, but MANTIS only uses the first character
<b>Consideration</b>	If you do not supply a value, MANTIS automatically assumes the space character.

---

**BEFORE**

<b>Description</b>	<i>Optional.</i> Specifies that unpadding occur only on the left-hand side of the variable (delete the leading pad characters).
--------------------	---

---

**AFTER**

<b>Description</b>	<i>Optional.</i> Specifies that unpadding occur only on the right-hand side of the variable (delete trailing characters). This is the default value.
--------------------	--

ALL

**Description**     *Optional.* Indicates that unpadding occurs on both sides of the variable (delete leading and trailing pad characters).

**General consideration**

See also “PAD” on page 346 and “POINT” on page 368.

**Examples**

The following examples of code and its results show how the PAD statement can be used with various symbols on either side of an expression. In the following examples:

```
00010 TEXT A(20),B(20)
00020 A="*****ABC*****"
00030 B="123"
```

Example	Results	Comments
UNPAD A " "	"*****ABC"	Default AFTER.
UNPAD A BEFORE	"ABC*****"	
UNPAD A ALL	"ABC"	
UNPAD B	"123"	Default " " AFTER.
UNPAD B " " BEFORE	"123"	No leading blanks.
UNPAD B ALL	"123"	
A="ABC-----++++++"	"ABC-----++++++"	Size=20.
UNPAD A(10,13)"-"	"ABC-----++++++"	Size=16.
A="    JOE B. JACKSON    "		Does not remove
UNPAD A	"JOE B. JACKSON"	middle blanks.

- ◆ The following example shows UNPAD being used to determine if a text field is all blanks or empty:

```
00010 UNPAD CLIENT_NAME
00020 IF CLIENT_NAME= " "
.
.
.
00050 END
```

- ◆ The following example shows UNPAD being used to determine if a text field is all zeros or empty:

```
00010 UNPAD CLIENT_NUMBER "0 "
00020 IF CLIENT_NUMBER= " "
.
.
.
00050 END
```

## UNTIL-END

The UNTIL-END statement executes a block of statements repeatedly until a specified condition becomes true. MANTIS executes the block of statements in the range of the UNTIL once before it tests the condition.

### UNTIL *expression*

.  
.  
.  
*statements*

.  
END

### *expression*

- Description** *Required.* Specifies the condition that ends the execution of the UNTIL loop.
- Format** Arithmetic or relational expression that evaluates to either TRUE (nonzero) or FALSE (zero)

### General considerations

- ◆ UNTIL-END executes the body (*statements*) at least once. If you need to omit execution of the body when the initial condition is false, use the WHILE-END statement.
- ◆ The UNTIL statement *expression* is not evaluated until *statements* execute the first time.
- ◆ You can use BREAK to terminate the loop or NEXT to perform the test and the next iteration.
- ◆ Each of the statements UNTIL and END must appear on a line by itself. Only a comment (separated by a colon) can follow the UNTIL expression.
- ◆ See also “BREAK” on page 136, “FOR-END” on page 226, “IF-ELSE-END” on page 274, “NEXT” on page 335, “WHEN-END” on page 508, and “WHILE-END” on page 510.

### Example

The following example shows how the UNTIL-END statement is used to set up a condition that is tested until true:

```
00010 SEED
00020 HEAD "RANDOM NUMBERS --"
00025 '"Press CANCEL to stop, ENTER to continue"
00030 UNTIL KEY="CANCEL"
00040 .SHOW RND(10)
00050 .WAIT
00060 END
```

---

# UPDATE

The UPDATE statement replaces a record on a MANTIS file, an external file, a personal computer file, an RDM logical view, or a TOTAL DBMS view file with an updated (altered) record.

## UPDATE (External file)

---

**UPDATE** *file-name* [LEVEL=*n*]

---

---

### *file-name*

<b>Description</b>	<i>Required.</i> Specifies the name (as defined in a previously executed ACCESS statement) of an external file where you want to update a record.
<b>Format</b>	A MANTIS symbolic name (see “ <a href="#">Symbolic names</a> ” on page 24)

---

### LEVEL=*n*

<b>Description</b>	<i>Optional.</i> Specifies the buffer number that contains the record you want to update.
<b>Default</b>	1
<b>Format</b>	Arithmetic expression that evaluates to a value in the range 1 through <i>m</i> , where <i>m</i> is the maximum buffer number, as defined in the corresponding ACCESS statement

### Considerations

- ◆ Only specify LEVEL=*n* when the *file-name* has buffers defined.
- ◆ MANTIS uses only the integer portion of *n*.

## General considerations

- ◆ An external file open is issued on the first DELETE, GET, INSERT, or UPDATE (when required).
- ◆ MANTIS replaces only the data elements in the record that are defined in the External file view. You do not need to read the record before you update it. Before you issue the UPDATE statement, make sure that the data elements represented through the external file view do in fact contain the right values.
- ◆ You cannot change a *key* field in a file record before an update; instead, delete the existing record and insert the new record with an altered key.
- ◆ For INDEXED files, the contents of key data elements identify the record to be updated.
- ◆ For SEQUENTIAL files, the contents of the associated reference variable that contains the Relative Byte Address (RBA) identify the record to be updated.



- ◆ For NUMBERED files, the Relative Record Number (RRN) contained in the corresponding reference variable identifies the record to be updated.
- ◆ MANTIS returns a text string in the variable called *file-name* that reflects the status of the operation:

Returned text string	Description
"	The UPDATE was successful.
"LOCK" *	The password specified in the ACCESS statement for this file view is not valid for updates.
"ERROR" *	<p>MANTIS received an error status. Use the FSI function (see "FSI" on page 232) for more information. Possible causes:</p> <ul style="list-style-type: none"> <li>◆ A physical error occurred during record updating.</li> <li>◆ The record to be updated did not exist on the file.</li> <li>◆ For SEQUENTIAL, variable-length record files, you tried to change the record length.</li> <li>◆ The External file exit canceled the operation.</li> </ul>
"NOTOPEN" *	The external file is not open.

\* Returned only when TRAP is in effect for the file.

- ◆ If you perform an UPDATE without a GET...ENQUEUE, the system internally performs a GET...ENQUEUE prior to performing the UPDATE. No data values are moved to program variables.
- ◆ For extended external file status messages and Function Status Indicators (FSIs), see “[Extended status messages for MANTIS and external files](#)” on page 521.
- ◆ The External File Exit can affect this statement. See your Master User for details.
- ◆ See also “[DELETE](#)” on page 183, “[DELETE \(Personal computer file\)](#)” on page 194, “[FSI](#)” on page 232, “[GET](#)” on page 234, “[INSERT](#)” on page 277, and “[TRAP](#)” on page 469.

**Example**

The following example shows how the UPDATE statement replaces an external file record:

```
00020 ACCESS RECORD( "INDEX", "SERENDIPITY", 16 )
00030 SCREEN MAP( "INDEX" )
00040 CONVERSE MAP
00050 COUNTER=1
00060 WHILE MAP<>"CANCEL"AND COUNTER<17
00070 .WHEN INDICATOR(COUNTER)="U"
00080 ..UPDATE RECORD LEVEL=COUNTER
.
.
.
```

---

## UPDATE (MANTIS file)

---

**UPDATE** *file-name* [LEVEL=*n*]

---

---

### *file-name*

<b>Description</b>	<i>Required.</i> Specifies the name (as defined in a previously executed FILE statement) of an existing MANTIS file where you want to update a record.
<b>Format</b>	A MANTIS symbolic name (see “ <a href="#">Symbolic names</a> ” on page 24)

---

### LEVEL=*n*

<b>Description</b>	<i>Optional.</i> Specifies the buffer number that contains the record you want to update.
<b>Default</b>	1
<b>Format</b>	Arithmetic expression that evaluates to a value in the range 1 through <i>m</i> , where <i>m</i> is the maximum buffer number, as defined in the corresponding FILE statement

### Considerations

- ◆ Only specify LEVEL=*n* when the *file-name* has buffers defined.
- ◆ MANTIS uses only the integer portion of *n*.

General considerations

- ◆ You do not need to read the record before updating it.
- ◆ If you perform an UPDATE without a GET...ENQUEUE, the system internally performs a GET...ENQUEUE prior to performing the UPDATE. No data values are moved to program variables.
- ◆ The contents of key data elements identify the record to be updated.
- ◆ You cannot change a *key* field in a file record before an update; instead, delete the existing record and insert the new record with an altered key.
- ◆ MANTIS returns a text string, in the variable called *file-name*, that reflects the status of the operation:

Returned text string	Description
""	The UPDATE was successful.
"LOCK" *	The password specified in the FILE statement for this file view is not valid for updates.
"ERROR" *	MANTIS received an error status. Use the FSI function (see "FSI" on page 232) for more information. Possible causes: <ul style="list-style-type: none"><li>◆ A physical error occurred during record updating.</li><li>◆ The record to be updated did not exist on the file.</li><li>◆ The SETPRAY Exit canceled the operation.</li></ul>

\* Returned only when TRAP is in effect for the file.

- ◆ The Setpray Exit can affect this statement. See your Master User for details.
- ◆ For extended MANTIS file status messages and Function Status Indicators (FSIs), see "Extended status messages for MANTIS and external files" on page 521.
- ◆ See also "DELETE" on page 183, "FSI" on page 232, "GET" on page 234, "INSERT" on page 277, and "TRAP" on page 469.

**Example**

The following example shows how a record in a MANTIS file is replaced with an altered record:

```
.  
.  
00060 ..CONVERSE MAP  
00070 ..WHEN MAP="PF1"  
00080 ...INSERT RECORD  
00090 ..WHEN MAP="PF2"  
00100 ...DELETE RECORD  
00110 ..WHEN MAP="PF3"  
00120 ...UPDATE RECORD  
00130 ..END  
00140 ..GET RECORD  
00150 .END  
00170 EXIT
```

## UPDATE (Personal computer file)

UPDATE *file-name* [LEVEL=*n*]

---

### *file-name*

<b>Description</b>	<i>Required.</i> Specifies the name (as defined in a previously executed ACCESS statement) of a file where you want to update a record.
<b>Format</b>	A MANTIS symbolic name (see “ <a href="#">Symbolic names</a> ” on page 24)

---

### LEVEL=*n*

<b>Description</b>	<i>Optional.</i> Specifies the buffer number that contains the record you want to update.
<b>Default</b>	1
<b>Format</b>	Arithmetic expression that evaluates to a value in the range 1 through <i>m</i> , where <i>m</i> is the maximum buffer number, as defined in the corresponding ACCESS statement

### Considerations

- ◆ Only specify LEVEL=*n* when the *file-name* has buffers defined.
- ◆ MANTIS uses only the integer portion of *n*.

### General considerations

- ◆ MANTIS replaces only the data elements in the record that are defined in the file view. You do not need to read the record before you update it. Before you issue the UPDATE statement, you must make sure that the data elements represented through the file view do in fact contain the right values.
- ◆ For SEQUENTIAL files, the contents of the associated reference variable that contains the Relative Byte Address (RBA) identify the record to be updated.
- ◆ For NUMBERED files, the Relative Record Number (RRN) contained in the corresponding reference variable identifies the record to be updated.

- ◆ MANTIS returns a text string, in the variable called *file-name*, that reflects the status of the operation:

Returned text string	Description
" "	The UPDATE was successful.
"LOCK" *	The password specified in the ACCESS statement for this file view is not valid for updates.
"ERROR" *	<p>MANTIS received an error status. Use the FSI function (see "FSI" on page 232) for more information. Possible causes:</p> <ul style="list-style-type: none"> <li>◆ A physical error occurred during record updating.</li> <li>◆ The record to be updated did not exist on the file.</li> <li>◆ You tried to change the record length of a SEQUENTIAL file record.</li> </ul>

\* Returned only when TRAP is in effect for the file.

- ◆ For extended personal computer file status messages and Function Status Indicators (FSIs), see "Extended status messages for MANTIS and external files" on page 521.
- ◆ The computer user can access files only residing on his or her personal computer.
- ◆ See also "DELETE" on page 183, "FSI" on page 232, "GET" on page 234, "INSERT" on page 277, and "TRAP" on page 469.

### Example

The following example shows how a personal computer file record is replaced by the UPDATE statement:

```

00020 ACCESS RECORD("INDEX", "SERENDIPITY", 16)
00030 SCREEN MAP("INDEX")
00040 CONVERSE MAP
00050 COUNTER=1
00060 WHILE MAP<>"CANCEL" AND COUNTER<17
00070 .WHEN INDICATOR(COUNTER)="U"
00080 ..UPDATE RECORD LEVEL=COUNTER
.
.
```

## UPDATE (RDM logical view)

UPDATE *view-name* [LEVEL=*n*]

### *view-name*

<b>Description</b>	<i>Required.</i> Specifies the symbolic name (as defined in a previously executed VIEW statement) of the logical view where you want to update a record.
<b>Format</b>	A MANTIS symbolic name (see “ <a href="#">Symbolic names</a> ” on page 24)

### LEVEL=*n*

<b>Description</b>	<i>Optional.</i> Specifies the buffer number that contains the record you want to update.
<b>Default</b>	1
<b>Format</b>	Arithmetic expression that evaluates to a value in the range 1 through <i>m</i> , where <i>m</i> is the maximum buffer number, as defined in the corresponding VIEW statement

### Considerations

- ◆ Only specify LEVEL=*n* when the *file-name* has buffers defined.
- ◆ MANTIS uses only the integer portion of *n*.

### General considerations

- ◆ If you try to alter the logical view key, you have actually requested a repositioning of the logical view, not a modification of the current record. To change a logical view key, you must first delete the old record, then insert a new one.
- ◆ Because RDM records might not be uniquely keyed, you *must* establish your current record position in a view (by reading the record) before you execute an UPDATE.



- ◆ MANTIS returns a text string, in the variable called *view-name*, that reflects the status of the operation:

Returned text string	Description
""	The UPDATE was successful.
"LOCK" *	You do not have permission to update logical records in the logical view.
"NOTFOUND" *	The variable-entry chain set with the requested key doesn't exist.
"ERROR" *	<p>MANTIS received an error status. Use the FSI function (see "FSI" on page 232) for more information. Possible causes:</p> <ul style="list-style-type: none"> <li>◆ An RDM error occurred during database access.</li> <li>◆ You tried to perform an invalid function for the user view.</li> </ul>

\* Returned only when TRAP is in effect for the file.

- ◆ RDM logical view update returns three status indicators to the application program that indicate processing results—FSI, ASI, VSI. FSI indicates the success or failure of your command. ASI indicates the status of each field in the logical record. VSI indicates the highest field status within the logical record. For a complete discussion of these status indicators, see "Status functions" on page 517.
- ◆ If you perform an UPDATE without a GET...ENQUEUE, the system internally performs a GET...ENQUEUE prior to performing the UPDATE.
- ◆ Use the GET...ENQUEUE prior to using the UPDATE function when computing a new value for a logical record (that is incrementing a counter). If you are simply using the UPDATE to place a value in a logical record, it is not necessary to issue a GET...ENQUEUE statement that is not dependent on the value already present.
- ◆ If TRAP is not in effect and the update cannot be applied because of a failure status from SUPRA RDM, MANTIS automatically issues a RESET. If TRAP is in effect and the program does not issue a RESET when "ERROR" is returned, then it is possible that MANTIS does only part of the modification.

- ◆ Your DBA can disallow updates. If so, MANTIS returns the “LOCK” status if TRAP is in effect. If TRAP is not in effect, MANTIS displays a message and halts execution.
- ◆ If you have issued an UPDATE since the last terminal I/O, MANTIS automatically issues a COMMIT prior to any terminal I/O (when COMMIT ON is in effect).
- ◆ See also “**ASI**” on page 93, “**COMMIT**” on page 149, “**DELETE**” on page 183, “**FSI**” on page 232, “**GET**” on page 234, “**INSERT**” on page 277, “**TRAP**” on page 469, and “**VSI**” on page 505.

**Example**

The following example shows how the UPDATE statement replaces an RDM logical view record:

```
00010 VIEW CUSTOMER("CUST")
00020 SCREEN MAP("CUST_UPDATE")
00030 SHOW"ENTER CUSTOMER NUMBER:"
00040 OBTAIN CUST_NO
00050 GET CUSTOMER(CUST_NO)
00060 IF CUSTOMER="FOUND"
00070 .INPUT_OK=FALSE
00080 .UNTIL INPUT_OK
00090 ..CONVERSE MAP
00100 ..DO EDIT_INPUT
00125 ..WHEN EDIT_OK
00130 ...INPUT_OK=TRUE
00135 ..END
00140 .END
00150 .UPDATE CUSTOMER
00160 .SHOW"CUSTOMER INFORMATION UPDATED"
.
.
.
```

## UPDATE (TOTAL file view)

---

**UPDATE** *file-name* [LEVEL=*n*]

---

### *file-name*

**Description**     *Required.* Specifies the name (as defined in a previously executed TOTAL statement) of TOTAL view containing the record you want to update.

**Format**             A MANTIS symbolic name (see “[Symbolic names](#)” on page 24)

---

### LEVEL=*n*

**Description**     *Optional.* Specifies the buffer number that contains the record you want to update.

**Default**             1

**Format**             Arithmetic expression that evaluates to a value in the range 1 through *m*, where *m* is the maximum buffer number, as defined in the corresponding TOTAL statement

### Considerations

- ◆ Only specify LEVEL=*n* when the *file-name* has buffers defined.
- ◆ MANTIS uses only the integer portion of *n*.

### General considerations

- ◆ A TOTAL file open is issued on the first DELETE, GET, INSERT, or UPDATE.
- ◆ MANTIS merges data residing in the specified variables with the existing record and rewrites the entire record to the database.
- ◆ When MANTIS performs an update for a TOTAL view, every field in the view must have user-specified data, or the update overlays existing data with blanks or zeros.
- ◆ If you change any keys in a variable-entry file record, MANTIS places the updated record in the file view associated with the new key values.

- ◆ You do *not* need to read the record before you update it. MANTIS always obtains exclusive control of the record for the current task before requesting TOTAL to update it.
- ◆ To ensure you have the most current data, you can first issue the GET statement with the ENQUEUE parameter (see GET).
- ◆ MANTIS returns a text string, in the variable called *file-name*, that reflects the status of the operation:

Returned text string	Description
""	The UPDATE was successful.
"NOTFOUND" *	The variable-entry chain set with the requested key does not exist.
"LOCK" *	The password specified in the TOTAL statement is invalid.
"NOTOPEN" *	The TOTAL view is not open.
"NOTAVAL" *	The TOTAL file or view is not open.

\* Returned only when TRAP is in effect for the file.

- ◆ See also "DELETE" on page 183, "GET" on page 234, "INSERT" on page 277, and "TRAP" on page 469.

**Example**

The following example shows how the UPDATE statement replaces a TOTAL DBMS record:

```
00020 TOTAL BILL("BOM", "ASSEMBLY", 8)
.
.
.
00100 UPDATE BILL LEVEL=BUFFER
```

---

# UPPERCASE

The UPPERCASE function converts a text string into uppercase.

---

## UPPERCASE(*t*)

---

*t*

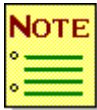
**Description**     *Required.* Specifies the text expression that you wish to convert to uppercase.

**Consideration** Specifies any valid text expression.

### General considerations

- ◆ Data from terminals can be automatically uppercased by field or terminal attributes, or by the TP monitor.
- ◆ UPPERCASE can be required to perform case-insensitive comparisons.

This statement can be affected by the TRCODE option in the Customization Table. See your Master User for details.




---

Uppercase translation can be modified by your System Administrator for your native language.

---

- ◆ The translation table used depends upon the current LANGUAGE setting.
- ◆ UPPERCASE can also be set on the Full Screen Edit Profile screen for the duration of the correct edit session. Refer to *MANTIS Program Design and Editing, OS/390, VSE/ESA*, P39-5013.
- ◆ See also “Text considerations” on page 22 and “LOWERCASE” on page 321.

Example

Example	Results	Comments
UPPERCASE ( "abc" )	"ABC"	
UPPERCASE ( "aBc % 123" )	"ABC % 123"	
UPPERCASE ( "â€" )	"Â€"	Depends on your language setting and translation table. See your Master User.

The following example shows how the UPPERCASE function is used to compare the content of two text strings, without regard to case:

```
00110 IF UPPERCASE (T1)=UPPERCASE(T2)
00120 .SHOW "Name match, record already exists"
00130 END
```

# USAGE

The USAGE command determines where a symbolic name appears in a program. MANTIS searches each statement within a given range for the symbolic name you specify.

---

**USAGE** *name* [,*n1*[,*n2*] ]

---



---

## *name*

<b>Description</b>	<i>Required.</i> Specifies the name of the object of the search.
<b>Format</b>	A MANTIS symbolic name (see “ <a href="#">Symbolic names</a> ” on page 24)

---

## *n1*

<b>Description</b>	<i>Optional.</i> Specifies the first statement number of the search.
<b>Default</b>	First line in your program
<b>Format</b>	Arithmetic expressions that evaluate to a value in the range of 0–30000
<b>Consideration</b>	MANTIS uses only the integer portion of <i>n</i> .

---

## *n2*

<b>Description</b>	<i>Optional.</i> Specifies how many occurrences of the symbolic name you want to display.
<b>Default</b>	Number of lines available on your terminal
<b>Format</b>	Arithmetic expressions that evaluate to a value in the range of 0–30000
<b>Consideration</b>	MANTIS uses only the integer portion of <i>n2</i> .

## General considerations

- ◆ You can terminate a USAGE listing at any time by typing KILL.
- ◆ USAGE is only valid in the MANTIS Line Editor. Use the FIND or RFIND commands in the Full Screen Editor.
- ◆ USAGE works only on full symbolic names. You cannot look for partial names, literals, or reserved words.

## Example

The following example shows how the USAGE command determines where a symbolic name will appear in a program:

```
USAGE REC1
```

```
30 FILE REC1 ( "INDEX" , "SERENDIPITY" )  
40 WHILE REC1<>"END"  
50 .GET REC1  
.  
.  
.
```



---

# USER

The USER function returns a text string identifying the current user name.

---

## USER

---

### General consideration

See also “PASSWORD” on page 349, “PRINTER (Function)” on page 370, “PRINTER (Statement)” on page 371, “TERMINAL” on page 455, and “TERMSIZE” on page 456.

### Example

Example	Results	Comments
USER	"NANCYJONES"	
USER ( 1 , 5 )	"NANCY"	Substringing permitted.

# USERWORDS

The USERWORDS function returns the number of MANTIS symbolic names currently in use.

## USERWORDS

### General considerations

- ◆ A limit of 2048 user words (symbolic names) exists for a single program. If you are nearing this limit, you should reduce the number of user words by using automapping, by reorganizing your program to use internal or external routines, or both.
- ◆ You can use this function to monitor your current usage. To get an accurate count, you should execute all of the complex statements (FILE, SCREEN, etc.).
- ◆ TEXT, BIG, SMALL, DBCS, KANJI, ENTRY, PROGRAM, and LET statements define new userwords. FILE, ACCESS, TOTAL, VIEW, SCREEN, and INTERFACE statements define the *name* on the statement. During execution, they implicitly define symbolic names for their elements.
- ◆ While in programming mode, RUN (without a line number), and SAVE or REPLACE (for an unbound program) will discard any implicitly defined symbolic names.
- ◆ See also “DATAFREE” on page 176 and “PROGFREE” on page 372.

### Example

The following example shows how the USERWORDS function returns the number of MANTIS symbolic names currently in use:

Example	Results	Comments
USERWORDS	0	For an empty program.
USERWORDS	65	For a sample program.

---

# VALUE

The VALUE function returns the numeric value of a text expression.

---

## VALUE(*t*)

---

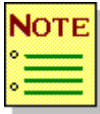
*t*

**Description**     *Required.* Specifies a text expression to convert to a numeric value.

**Consideration** Any valid text expression.

### General considerations

- ◆ The text string is evaluated using the MANTIS number set as specified in Numeric Considerations under “**Symbolic names**” on page 24.



---

If the text string contains the character E immediately prior to numeric digits, then the numeric value returned from the function might not be what you expect, because the E signifies scientific numeric representation, exponent, for example 12E38. If the text string contains the character E, this function must be used with caution.

---

- ◆ Characters other than 0–9, E, “+”, “-”, and “.” are ignored. No operations are done on the string.
- ◆ If one decimal point is in the string, it is used. Multiple decimal points in the string are considered punctuation and are all ignored. (The decimal point is “.” or “,” as defined in your user profile.)
- ◆ See also “**TXT**” on page 473.

Example

Example	Results	Comments
VALUE ( " 39 " )	39	
VALUE ( "-40" )	-40	
VALUE (KEY)	3	If KEY="PF3"
VALUE (KEY)	0	If KEY="CANCEL "
VALUE ( "98.6" )	98.6	
VALUE ( "12.6.2002" )	1262002	Multiple decimals are treated as punctuation.
VALUE (NULL)	0	
VALUE ( "12E9" )	.12E11	E-notation.
VALUE ( "5 Slippers" )	5	
VALUE ( "Oxford 4E-10" )	.4E-9	E-notation.
VALUE ( "612-2300" )	-6122300	
VALUE ( "5+7" )	57	No operations are done on the string.

# VIEW

The VIEW statement specifies an RDM logical view. MANTIS retrieves the view from the SUPRA Directory. If the view is not known to RDM or not authorized for your use, MANTIS returns an error message and halts execution. If the view is valid, MANTIS opens it and establishes the MANTIS variables as they are defined in the SUPRA Directory with the following exceptions:

- ◆ MANTIS converts all hyphens (-) in logical view field names to underscores (\_) in MANTIS variable names and vice versa.
- ◆ The characters \$ and # are invalid in MANTIS. If the logical view has field names with these characters in them, MANTIS returns an error message and halts execution.

VIEW

```
name1(view - name1[,PREFIX][,n1][,SELECT (f1list1[,f1list2...]))  
[, name2 (view - name2 [,PREFIX][,n2][,SELECT (f2list1[,f2list2...]))...]  
...  
ON[(user - id[,password])]  
OFF
```

## name

Description	<i>Required.</i> Specifies the name that you use to refer to the logical view in subsequent GET, UPDATE, INSERT, DELETE, TRAP, and MARK statements.
Format	A MANTIS symbolic name (see “Symbolic names” on page 24)
Consideration	When the symbolic name is previously defined, MANTIS bypasses this definition.

## view-name

Description	<i>Required.</i> Specifies the name of an existing logical view as defined in the SUPRA Directory.
Format	1–30 character text expression that evaluates to a valid RDM logical view name
Consideration	This parameter is translated to uppercase upon execution of your program.

---

## PREFIX

**Description** *Optional.* Indicates that MANTIS place the symbolic name and an underscore before all field names associated with this logical view. If, for example, you code:

```
10 VIEW BIN("BOTTLES",PREFIX)
```

and the logical view BOTTLES has a field named VOLUME, the program would refer to that field now as BIN\_VOLUME.

**Format** Must be coded exactly as shown

**Consideration** See the PREFIX considerations under “**FILE**” on page 222.

---

## *n*

**Description** *Optional.* Indicates how many buffers MANTIS should allocate. Each buffer contains data accessible through the logical view from a single logical view record.

**Default** 1

**Format** Arithmetic expression that evaluates to a value in the range 1–255

**Consideration** When you use the *n* parameter, any GET, INSERT, UPDATE, DELETE, and MARK statements must contain the LEVEL=*n* option to tell MANTIS which buffer to process.

---

## SELECT(*f1list1*,...,*fnlistn*)

**Description** *Optional.* Specifies a list (or a series of lists) of logical view fields that this MANTIS program uses. MANTIS transmits only the fields specified to or from RDM.

**Format** Each list is a text expression, up to 254 characters in length, that contains the logical view field names separated by a comma (,). Each field name must conform to naming conventions established by RDM and to all restrictions specified in this document.

### Considerations

- ◆ MANTIS converts any underscores (\_) to hyphens (-) in the requests made to RDM.
- ◆ This parameter is translated to uppercase upon execution of your program.

---

**ON**

**Description**     *Optional.* Forces MANTIS to sign on to SUPRA RDM.

**Consideration** Use the VIEW ON parameter to force a different user or password to be used for SUPRA RDM. This parameter is needed when interface programs that have signed off to RDM are called.

---

***user-id***

**Description**     *Optional.* Indicates the RDM user to which you want to sign on.

**Default**            Current MANTIS user

---

***password***

**Description**     *Optional.* Specifies the password associated with the specified user ID.

**Default**            Current password

---

**OFF**

**Description**     *Optional.* Forces MANTIS to sign off from SUPRA RDM.

**Consideration** Use the VIEW OFF parameter when interface programs require the task to be signed off prior to the CALL.

## General considerations

- ◆ The SUPRA Directory must authorize access to a logical view before a MANTIS user can execute the VIEW statement.
- ◆ If you fail to include required fields in the logical view, you cannot perform inserts and some updates on the logical view.
- ◆ Modifying the key order in the VIEW statement SELECT option could adversely affect performance. Your DBA has defined the key order on the SUPRA Directory to maximize performance.
- ◆ A VIEW statement can appear anywhere in your program. Must be executed before the first GET, UPDATE, INSERT, or DELETE that references it.
- ◆ The symbolic name you specify on a VIEW statement becomes the name of the user view (a subset of the logical view). Therefore, if you use the same symbolic name on more than one VIEW statement (at different DO levels), the VIEW statements must refer to the same logical view and must have an identical SELECT list. If the VIEW statements are different, then they must each have a different MANTIS symbolic name.
- ◆ MANTIS normally signs on upon first access to RDM and signs off in the **Sign on As Another User** Facility Program.
- ◆ Arguments (except for passwords) for the VIEW statement are translated to uppercase upon execution of your program.
- ◆ See also “**ASI**” on page 93, “**DELETE**” on page 183, “**FSI**” on page 232, “**INSERT**” on page 277, “**RELEASE (Function)**” on page 378, “**RELEASE (Statement)**” on page 380, “**TRAP**” on page 469, “**UPDATE**” on page 479, and “**VSI**” on page 505.

## Examples

- ◆ The following example shows how the VIEW statement identifies an RDM logical view:

```
00010 VIEW CUSTOMER( "CUST" )
00020 VIEW CUST_ITEM( "CUST_ITEM",10,SELECT( "CUST_NO,ITEM_NUM",
00030 "QUANTITY_ON_ORDER" ) )
```
- ◆ The following example shows how the VIEW statement may be used to sign on to RDM as a different user:

```
00010 VIEW ON( "PRODUSER", "X" )
```



# VSI

The VSI function indicates the highest field status (ASI) for the last operation on a view.

**VSI(*view-name*)**

***view-name***

- Description**
- Required.* Specifies the name of the logical view.
- Format**
- A MANTIS symbolic name (see “**Symbolic names**” on page 24)

**General considerations**

- ◆ Validity Status Indicators (VSIs) reflect the overall validity of the user view record you used in your last request. See “**RDM status functions**” on page 517 for more details.
- ◆ See also “**ASI**” on page 93, “**DELETE**” on page 183, “**FSI**” on page 232, “**GET**” on page 234, “**INSERT**” on page 277, “**TRAP**” on page 469, “**UPDATE**” on page 479, and “**VSI**” on page 505.

**Example**

Example	Results	Comments
VSI(PARTS)	"CHANGED"	

The following example shows how the VSI function can be used to test a condition:

```
00020 VIEW PARTS("PARTS_ON_ORDER")
.
.
.
00100 GET PARTS
00110 IF VSI(PARTS)="CHANGED"
.
.
.
```

## WAIT

The WAIT statement temporarily suspends execution of a program. You generally use the WAIT statement to display unformatted data (from a SHOW statement) on the screen until you press a key to continue execution. You must use a WAIT statement to display SHOWS when you execute a program from a menu.

---

### WAIT

---

#### General considerations

- ◆ To view data, you must include a WAIT or OBTAIN statement after a SHOW and before PROMPT, CONVERSE, and STOP.
- ◆ If you do not execute a WAIT, the data from SHOWs are only displayed when the screen full of data is available for the terminal, or an OBTAIN is done. Uncontinued SHOWS—those ending with a semicolon (;)—can be displayed on the message line of a subsequent CONVERSE.
- ◆ You can include a WAIT in your program for debugging purposes. You can enter KILL to stop execution or press ENTER (or any PF key) to continue without having to respecify RUN line number (see Example 2).
- ◆ See also “KEY” on page 301, “OBTAIN” on page 341, and “SHOW” on page 400.
- ◆ If you code a WAIT statement directly below a CONVERSE, the WAIT will be included as part of the CONVERSE. This processing is the same for OBTAIN (see “OBTAIN” on page 341). If you want a WAIT to actually occur in this situation, you need to code two WAIT statements together (e.g., WAIT:WAIT). The first will be executed along with the CONVERSE and the second will be executed by itself. The following example illustrates this concept:

```
00010  ENTRY WAIT_TEST
00020  SCREEN MAP( "test" )
00030  CONVERSE MAP
00040  WAIT:WAIT
00050  EXIT
```

## Examples

- ◆ The following example shows how the WAIT statement suspends execution of a program, so that unformatted data can be displayed:

```

00010 .WHILE MAP<>"CANCEL"
00020 ..GET REC(CUST_NUMBER)EQUAL
00030 ..SHOW REC:WAIT
00040 ..IF REC="FOUND"
00050 ...MESSAGE=" 'PF1' TO UPDATE      'PF2' TO CANCEL"
00060 ...CONVERSE MAP
00070 ...WHEN MAP="PF1"
00080 ....UPDATE REC
00090 ....MSG="UPDATE COMPLETE"
00100 ...WHEN MAP="PF3"
00110 ....MSG="MAINTENANCE CANCELLED AT USER'S REQUEST"
00120 ...END
00130 ..ELSE
00140 ...MSG="CUSTOMER NOT FOUND"
00150 ..END
00160 ..CLEAR MAP
00170 ..MESSAGE=MSG
00180 ..CONVERSE MAP
00190 .END

```

- ◆ The following example shows lines inserted for debugging purposes:

```

1000 GET REC
1010 SHOW "@1010",REC,FIELD1,FIELD2,:WAIT
...
3000 ENTRY SUB(A)
3010 IF DEBUGGING
3020 .SHOW"@ CHECKPOINT3 - IN SUB - Argument is";A:WAIT
3030 END

```

---

# WHEN-END

The WHEN-END statement executes a block of statements when a specified condition is TRUE. MANTIS performs the test before executing the block. If the condition is false, or after executing the block, execution proceeds to the next WHEN.

---

```
WHEN expression
    .statements
[WHEN expression
    .statements]
...
END
```

---

---

## *expression*

**Description**     *Required.* Specifies the condition required before MANTIS executes the block of statements.

**Format**            Arithmetic or relational expression that evaluates to a value of TRUE (nonzero) or FALSE (zero)

**General considerations**

- ◆ MANTIS evaluates every WHEN statement in a program, even if an earlier condition is met. A BREAK statement can cause execution to pass to the statement after the END. The conditions are independent and may not be mutually exclusive (see example below).
- ◆ A BREAK statement causes execution to pass to the statement after the END.
- ◆ A NEXT statement causes execution to pass to the next WHEN *condition* from the middle of a statement block.
- ◆ Both the WHEN and the END statements must appear on a line by themselves. MANTIS ignores any additional statements coded on the end of a WHEN or END statement (and separated by a colon).
- ◆ WHEN-END structures cannot be directly nested, unless separated by an intervening IF-ELSE-END, FOR-END, UNTIL-END, or WHILE-END. Otherwise, the nesting level would be ambiguous.
- ◆ See also “BREAK” on page 136, “FOR-END” on page 226, “IF-ELSE-END” on page 274, “NEXT” on page 335, “WHEN-END” on page 508, and “UNTIL-END” on page 478.

**Example**

The following example shows how the WHEN-END statement executes a particular block of statements when a particular condition is met:

```
00010 ENTRY INDEX
00020 .FILE RECORD( "INDEX", "SERENDIPITY" )
00025 .FILE RECORD1( "INDEX", "SERENDIPITY" )
00030 .SCREEN MAP( "INDEX" )
00040 .GET RECORD
00050 .WHILE RECORD<>"END"AND MAP<>"CANCEL"
00060 ..CONVERSE MAP
00063 ..WHEN MAP="CANCEL"
00066 ...STOP
00070 ..WHEN ACTION="DELETE"
00080 ...DELETE RECORD
00090 ..WHEN ACTION="KEYCHANGE"
00100 ...DELETE RECORD1(OLDKEY) ALL
00110 ..WHEN ACTION="KEYCHANGE" OR ACTION="INSERT"
00120 ...INSERT RECORD
00130 ..WHEN MAP<>"CANCEL"
00140 ...GET RECORD
00150 ..END
00160 .END
00170 EXIT
```

## WHILE-END

The WHILE-END statement executes a block of statements repeatedly while a specified condition is true. If the condition is FALSE, MANTIS terminates the WHILE and executes the statement after END. If the condition is TRUE, MANTIS executes the statements within the WHILE range and then reevaluates the expression.

---

**WHILE** *expression*  
    *.statements*  
**END**

---

---

### *expression*

**Description**     *Required.* Specifies the condition that must exist while MANTIS executes the block of statements.

**Format**            Arithmetic or relational expression that evaluates to a value of TRUE (nonzero) or FALSE (zero)

### **General considerations**

- ◆ WHILE-END will **not** execute the body (*statements*) if the expression is initially FALSE (zero). If you need to execute the body once, regardless of the initial condition, use the UNTIL-END statement.
- ◆ You can use BREAK to terminate the loop or NEXT to perform the test and the next iteration.
- ◆ Both the WHILE and the END statements must appear on a line by themselves. MANTIS ignores any additional statements coded on the end of a WHILE or an END statement (and separated by a colon).
- ◆ See also “**BREAK**” on page 136, “**FOR-END**” on page 226, “**IF-ELSE-END**” on page 274, “**NEXT**” on page 335, “**WHEN-END**” on page 508, and “**UNTIL-END**” on page 478.

**Example**

The following example shows how the WHILE-END statement is used to execute a block of code as long as a specific condition is true:

```
00010 ENTRY INDEX
00020 .FILE RECORD( "INDEX", "SERENDIPITY" )
00030 .SCREEN MAP( "INDEX" )
00040 .GET RECORD
00050 .WHILE RECORD<>"END"AND MAP<>"CANCEL"
00060 ..CONVERSE MAP
00070 ..WHEN MAP="PF1"
00080 ...INSERT RECORD
00090 ..WHEN MAP="PF2"
00100 ...DELETE RECORD
00110 ..WHEN MAP="PF3"
00120 ...UPDATE RECORD
00130 ..END
00140 ..GET RECORD
00150 .END
00170 EXIT
```

# ZERO

The ZERO function returns the value zero.

## ZERO

### General considerations

- ◆ The following expressions are equivalent:  

NATION\_COUNTER=ZERO  
NATION\_COUNTER=0
- ◆ See also “FALSE” on page 221, “NULL” on page 338, and “TRUE” on page 472.

**Example** The following example shows how the ZERO function returns the value of zero:

Example	Results	Comments
ZERO	0	



# A

## Dissimilarity debugging

Dissimilarity errors occur in MANTIS when processing a complex statement (e.g., SCREEN, FILE, or ACCESS). This appendix defines the types of dissimilarity and suggests how you can locate and correct these errors.

The following types of dissimilarity exist in MANTIS:

- ◆ **TYPE.** Refers to BIG, SMALL, DBCS, KANJI, and TEXT definitions.
- ◆ **DIMENSION.** Can occur for the following reasons:
  - When identically named text fields executed by ACCESS, FILE, TOTAL or VIEW statements are too small.
  - When an array has been defined on a layout or repeats appear on a screen.
  - When repeats on a screen design have created an arrayed element.
  - When the LEVEL parameter on a FILE, INTERFACE, or TOTAL-related statement has created an array.

MANTIS displays the name of the field in question in the error message. Use the FIND command to locate any previous definitions. Note that the previous definition can be by another complex statement, in which case the FIND (Full Screen Editor only) or the USAGE command (Line Editor only) does not display it. Also, if the variable was passed to the routine through an ENTRY statement, you may need to work backwards into the calling routine to resolve the problem. Finally, you verify that the field definition in the design or view and, if applicable, the number of levels specified, are correct.

Use the SIZE function (operands "DIM", "MAX", 1, and 2) to determine currently defined dimensions. See also "[SIZE](#)" on page 404 for more information.

# B

## MANTIS reserved words

This appendix lists reserved words as they were added with each new release of MANTIS. It is organized with the most recent additions listed last. These reserved words also appear in “[Overview of MANTIS language](#)” on page 21.

### Release 4.0

ASI	EQUAL	PROGRAM	VSI
DOLEVEL	FSI	TERMSIZE	

### Release 4.2

BIND	DOWN	MODIFIED	SUBMIT
BREAK	EDIT	NULL	TO
BY	FOR	ORD	ULTRA
CHR	GO	PAD	UNPAD
CURSOR	KANJI	PERM	UP
DBPAGE	LANGUAGE	RETURN	WINDOW

**Release 5.2**

DBCS*	MIXM	NULL	SQLCA
EXEC_SQL	MIXMODE	NUMERIC	SQLDA
LOWERCASE	MIXT	SQLBIND	UPPERCASE
MIXD			

\* This reserved word is available with service level 5231 and above.

**Release 5.4**

BLOB	DECIMAL	INTEGER
------	---------	---------

**Release 5.5**

LUID

The following list contains Full Screen Editor commands. These are not reserved words, but they could require use of the escape character to eliminate ambiguity. Refer to *MANTIS Program Design and Editing, OS/390, VSE/ESA*, P39-5013, for details on the use of the escape character.

BOT	EC	LOC	RES
BOTTOM	ERRCODE	PRINT	RF
C	F	PROF	RFIND
CAN	FIND	PROFILE	RIGHT
CANCEL	L	RC	TOP
CHANGE	LEFT	RCHANGE	
CHG	LOCATE	RCHG	

# C

## Status functions

This appendix describes the meanings of several types of statuses that are returned by the status indicators FSI, (Function Status Indicators), ASI (Attribute Status Indicators, and VSI (Validity Status Indicators). Other sections describe extended status messages for internal and external files and views and corresponding message text you can receive while working in other environments.

### RDM status functions

RDM provides three status functions that can indicate the success or failure of a RDM logical view DELETE, GET, INSERT, or UPDATE. The tables in this appendix list and describe the indicators that these functions return.

- ◆ **FSI (Function Status Indicators).** See “[Function Status Indicators](#)” on page 518.
- ◆ **ASI (Attribute Status Indicators).** See “[Attribute Status Indicators](#)” on page 519.
- ◆ **VSI (Validity Status Indicators).** See “[Validity Status Indicators](#)” on page 520.

### Function Status Indicators

The FSI indicates the success or failure of your command. The ASI indicates the status of each field in the logical record. The VSI indicates the highest field status within the logical record. An appropriate error message appears with each indicator. Function Status Indicators appear in the following table:

MANTIS FSI value	RDM FSI value	Meaning
DATA	D	Data error. The logical record contains invalid data. Check the ASIs to find the field(s) containing the invalid value.
DUPLICATE	N	Failed due to an occurrence problem. This can be due to an INSERT duplicate found.
ERROR	F	This indicates a major error. Something could be wrong with the database, or you could have attempted to perform an illegal function on the user view.
GOOD	*	Successful GET or UPDATE.
NOTFOUND	N	Failed due to an occurrence problem. This can be due to a GET not found.
RESET	X	RESET recommended—During processing, function modifications were made to the database before MANTIS detected the error condition. Issue a RESET to restore the database.
RESTART	R	Restart request on COMMIT.
SECURITY	S	Security.
UNAVAILABLE	U	Unavailable resources.

### FSI example

Consider the following FSI example:

```
00020  VIEW CUSTOMERS( "NEW_CUSTOMERS" )
      .
      .
      .
00100  GET CUSTOMERS
00110  IF FSI( CUSTOMERS ) <> "GOOD"
      .
      .
      .
```

## Attribute Status Indicators

Attribute Status Indicators (ASI) reflect the status of each field defined in your logical view. ASIs have one-to-one mapping to each external field. MANTIS Attribute Status Indicators appear in the following table:

MANTIS ASI value	RDM ASI value	Meaning
CHANGED	C	Field value changed by another view.
DATA	V	The field contains an invalid value.
MISSING	-	The field is missing. It has a null value. (Valid for GET processing only.)
NEW	+	The field exists, but has changed since the last access. (Valid for GET processing only.)
SAME	=	The field exists and has not changed since the last access.

The following list shows the three ways to use ASIs:

1. When you issue a GET command, certain fields that are returned may not have a value. You can check this status (on fields that have not been altered) with the ASI.
2. If you receive an FSI indicating a data error, you can use the ASI to find which fields have illegal values.
3. When you issue an UPDATE command, another view may have updated a field value since your last GET. If this is the case, MANTIS doesn't perform your UPDATE. Use the ASI function to check this status and determine what course of action to take.

### ASI example

Consider the following ASI example:

```
00020  VIEW PARTS( "PARTS_ON_ORDER" )
      .
      .
      .
00100  GET PARTS
00110  IF ASI(PARTS,PART_NAME)="MISSING"
```

### Validity Status Indicators

Validity Status Indicators (VSI) reflect the overall validity of the user view record you used in your last request. For example, if an ASI returns a “DATA” status, the VSI is “DATA”. Check this indicator before checking each ASI. MANTIS Validity Status Indicators appear in the following table:

MANTIS VSI value	RDM VSI value	Meaning
CHANGED	C	Field value changed by another view.
DATA	V	At least one invalid ASI was returned.
MISSING	-	No invalid ASIs were returned, but at least one missing ASI was returned.
NEW	+	No invalid or missing ASIs were returned, but at least one field has changed since the last access.
SAME	=	No invalid, missing or new physical occurrences were returned for this RDM function.

The VSI enables you to determine if you need any additional processing of ASIs to correct invalid data or to fill in missing values.

### VSI example

Consider the following VSI example:

```
00020  VIEW PARTS ( "PARTS_ON_ORDER" )
      .
      .
      .
00100  GET PARTS
00110  IF VSI ( PARTS ) = "CHANGED"
      .
      .
      .
```



## Extended status messages for MANTIS and external files

MANTIS provides extended status messages as well as Function Status Indicators (FSIs) for internal (MANTIS) files, and external files (such as VSAM and PC CONTACT). The [table](#) under “[File status codes and messages](#)” on page 522 lists the various file statuses that are returned.

The MANTIS and external file statuses are returned when you perform a DELETE, GET, INSERT, or UPDATE. To obtain the FSI codes, use the following function:

```
FSI(filename[,msg])
```

This function returns GOOD, DUPLICATE, NOTFOUND, UNAVAILABLE, or ERROR, depending upon the external file status message left by the TP monitor. The following is a short example of how to use this function:

```
10  ACCESS X("TEST", "PSW")
15  TEXT MSG(40)
20  TRAP X ON
30  GET X FIRST
40  IF X="ERROR"
50  .IF FSI(X,MSG)="UNAVAILABLE"
60  ..SHOW"FILE TEST IS UNAVAILABLE"
70  .ELSE
80  ..SHOW"FILE TEST GET FAILED:STATUS="+MSG
90  .END
110 END
```

A SHOW MSG after an inquiry of FSI (that is ‘SHOW FSI(REC,MSG)’ after ‘IF FSI(REC,MSG)’...) returns the external name of the VSAM file and the external file status as returned from CICS.

The following table shows three levels of status messages. The MANTIS or external file status is returned in the symbolic name of the file to show the success of an operation. FSI status codes and messages are returned by the FSI function, and the message further explains the code.

File status codes and messages

FSI codes and meanings vary depending on your operating environment. The remaining three tables contain explanations of the FSI message texts for different environments:

- ◆ **CICS MANTIS.** See “CICS MANTIS FSI message text descriptions for internal and external files or views” on page 524.
- ◆ **MANTIS for batch.** See “MANTIS for batch FSI message text descriptions for internal and external files or views” on page 525.
- ◆ **PC CONTACT.** See “PC CONTACT FSI message text descriptions for internal and external files” on page 526.

MANTIS/external file status	FSI	FSI message
null ("")	GOOD	XXXXX DELETED RECORDS*
null ("")	GOOD	null ("")
FOUND	GOOD	null ("")
NEXT (UNKEYED GET)	GOOD	null ("")
NEXT (KEYED GET)	NOTFOUND	null ("")
NOTFOUND	NOTFOUND	null ("")
END	NOTFOUND	null ("")
DUPLICATE	DUPLICATE	DUPLICATE REC
ERROR	DUPLICATE	DUPLICATE KEY
NOT OPEN	UNAVAILABLE	NOTOPEN
ERROR	KEYLNTH ERR	NOTOPEN
NOT OPEN	UNAVAILABLE	UNAVAILABLE
ERROR	NOTFOUND	NOTFOUND
ERROR	ERROR	ILLOGICAL
ERROR	ERROR	INVREQ
ERROR	ERROR	FULL (NO SPACE)

\* Returned for a generic (ALL) DELETE request only

MANTIS/external file status	FSI	FSI message
ERROR	ERROR	LRECL INVALID
ERROR	ERROR	REMOTE INVREQ
ERROR	ERROR	SIZE
ERROR	ERROR	OPEN ERROR XXX YYY
ERROR	CANCELED	Any text added by the External file or SETPRAY exits
ERROR	ERROR	CONVERTER MISSING
ERROR	ERROR	I/O ERROR
ERROR	ERROR	INVALID ID
ERROR	ERROR	INVRESP
ERROR	ERROR	LOCKED
ERROR	ERROR	INVALID REPLY FOR <div><div>DELETE</div><div>FILE HANDSHAKE</div><div>GET</div><div>HANDSHAKE</div><div>INSERT</div><div>UPDATE</div></div> XXX
ERROR	ERROR	UNSUCCESSFUL <div><div>DELETE</div><div>FILE HANDSHAKE</div><div>GET</div><div>HANDSHAKE</div><div>INSERT</div><div>UPDATE</div></div>
None of the Above	None of the Above	Not Checked

CANCELED is for external files and means a request was canceled by a user exit. (There is not equivalent RDM code that is returned.)

## CICS MANTIS FSI message text descriptions for internal and external files or views

Message text	Meaning
CANCELED	Operation canceled by User exit.
DUPLICATE KEY	You are attempting to insert a record retrieved via an alternate index and with a nonunique key attribute. A record with the same key already exists.
DUPLICATE REC	A record with the same key already exists.
FULL	There is no space left for inserting records into the file.
ILLOGICAL XXX YYY	Internal error. This error occurs when the other error categories do not apply. XXX indicates the return code; YYY, the reason code. Refer to the <i>CICS/VS Application Programmer's Reference Guide—Command Level</i> for details.
INVREQ	MANTIS attempted an invalid request. Refer to the <i>CICS/VS Application Programmer's Reference Guide—Command Level</i> for details.
I/O ERROR XXX YYY	An error has occurred in attempting to access the file. XXX indicates the return code; YYY, the reason code. Refer to the <i>CICS/VS Application Programmer's Reference Guide—Command Level</i> for details.
LOCKED	Resource security check has failed. Refer to the <i>CICS/VS Application Programmer's Reference Guide—Command Level</i> (“NOTAUTH” status) for details.
LRECL INVALID	The record length specified in the profile is greater than the real record length. It should be less than or equal to the real record length.
NOTFOUND (IF MANTIS STATUS =“ERROR”)	Internal error. MANTIS attempted either to retrieve a record by key or positioned itself on the record, and when it attempted to perform an UPDATE or DELETE, the record was no longer there.
NOTOPEN	MANTIS attempted to perform a GET, INSERT, UPDATE, or DELETE on a file that was not open or disabled. Refer to CICS “NOTOPEN” or “DISABLED” statuses.
REMOTE INVREQ	MANTIS attempted an invalid request on a remote file. Refer to the <i>CICS/VS Application Programmer's Reference Guide—Command Level</i> for details.
SIZE	Internal error. Refer to the <i>CICS/VS Application Programmer's Reference Guide—Command Level</i> (“LENGERR” status) for details.
XXXXX DELETED RECORDS	MANTIS performed a generic DELETE request and XXXXX records were deleted.

## MANTIS for batch FSI message text descriptions for internal and external files or views

Message text	Meaning
DUPLICATE REC	A record with the same key already exists.
I/O ERROR XXX YYY	An error has occurred when MANTIS tried to access the file. XXX is the return code; YYY, the reason code. Refer to the appropriate VSAM messages and codes manual for details.
LRECL INVALID	The record length specified in the profile is greater than the real record length. It should be less than or equal to the real record length.
NOTFOUND (IF MANTIS STATUS = "ERROR")	When MANTIS attempted to perform an UPDATE or DELETE, the record did not exist.
NOTOPEN	Internal error. MANTIS attempted to perform a GET, INSERT, UPDATE, or DELETE on a file that was not open.
OPEN ERROR XXX YYY	XXX is the return code; YYY, the reason code. Refer to the appropriate VSAM messages and codes manual for an explanation.
UNAVAILABLE	There is a missing DLBL or DD statement for a file MANTIS is attempting to open.

PC CONTACT FSI message text descriptions for internal and external files

Message text	Meaning
CONVERTER MISSING	PC CONTACT was unable to locate its file conversion routine for the file.
I/O ERROR	An I/O error occurred while PC CONTACT was reading or writing a PC file.
INVALID REPLY FOR <div><div>DELETE</div><div>FILE HANDSHAKE</div><div>GET</div><div>HANDSHAKE</div><div>INSERT</div><div>UPDATE</div></div> XXX	MANTIS received a reply message (DELETE, INSERT, GET, or UPDATE ) record, file handshake or handshake that contains an unknown reply code. This can be caused by a logic error occurring between MANTIS and PC CONTACT.
INVALID ID	PC CONTACT detected an invalid record ID.
INVRESP	MANTIS received an unreadable reply message from PC CONTACT. This can be caused by: <ul style="list-style-type: none"><li>◆ PC CONTACT is not executing on the PC.</li><li>◆ A communication error destroyed the reply message.</li></ul>
NOTFOUND (IF MANTIS STATUS="ERROR")	PC CONTACT was unable to locate the PC file.
TRUNCATED	The PC file was longer than the length defined in the file profile, so MANTIS truncated the text field. (TRAP ON would cause a fault.)
UNSUCCESSFUL <div><div>DELETE</div><div>FILE HANDSHAKE</div><div>GET</div><div>HANDSHAKE</div><div>INSERT</div><div>UPDATE</div></div>	PC CONTACT was unable to: <ul style="list-style-type: none"><li>◆ Delete the record on the PC file.</li><li>◆ Open the PC file.</li><li>◆ Get the record from the PC file.</li><li>◆ Insert the record on the PC file.</li><li>◆ Update the record on the PC file.</li><li>◆ Establish communication with the PC.</li></ul>

# D

## Advanced programming techniques

This appendix presents advanced programming techniques that combine more than one statement. The following sections include information about External DO, program architecture, modularization, automatic mapping, debugging, application conversion, and VSAM deadlocks.

### External DO

The external DO statement allows a MANTIS program to transfer data and control of execution to another MANTIS program with an automatic return path to the next statement. If used optimally, external DO can improve the efficiency of execution and make it easier to maintain your applications.

External DO allows many programs to share common subroutines. Using external DO, you can set up programs that can be shared by other users, allowing them access to both internal and external subroutines. You can also link subroutines with external DO.

To achieve the best and most efficient use of external DO, there are a number of important factors to keep in mind while designing applications. This appendix explains how to use external DO.

- ◆ **Internal DO.** Allows a MANTIS program to transfer data and control of execution to a routine within the same program and return to the next statement following the DO. An internal routine may be passed variables as arguments but automatically has access to all variables defined in the running program.
- ◆ **External DO.** Allows a MANTIS program to transfer data and control of execution to a routine outside the same program and return to the next statement following the DO. An external routine has access only to those of the calling routine's variables that are passed to it as arguments.
- ◆ **CHAIN.** Allows a MANTIS program to transfer data and control of execution to another MANTIS program without an automatic return path.
- ◆ **CALL.** Allows a MANTIS program to transfer data and control of execution to a non-MANTIS program and automatically return to the next statement following the CALL.
- ◆ **PERFORM.** Allows a MANTIS program to transfer control of execution to a non-MANTIS program and optionally automatically return to the next statement. Also used to initiate a background task running a MANTIS or non-MANTIS program.
- ◆ **COMPONENT.** Identifies each component that can be assembled by the Compose action into an executable (composed) program. This allows a single version of the source to be included and tracked in multiple programs.



The following reserved words support external DO:

- ◆ **PROGRAM.** A statement that identifies and loads the external MANTIS programs that are to be invoked.
- ◆ **DOLEVEL.** Is a new built-in function that returns a value identifying the level that an external program is currently executing. The root program has a DOLEVEL of zero. With each External DO statement, the DOLEVEL value increases by one. With each EXIT from an external subroutine, the DOLEVEL decreases by one.
- ◆ **EXIT.** As a program editor command, EXIT allows you to exit to the calling routine in an immediate mode of execution. As a programming statement, EXIT continues to mark the end of a routine and to affect the return to the routine containing the DO.
- ◆ **RETURN.** Allows a program to return to the calling routine containing the DO from a point other than at the end (EXIT) of the called routine.

Any program can use both internal and external DOs. Internal routines are defined on ENTRY statements within the calling program. External routines are defined in PROGRAM statements within the calling program.

## Using external DO

The program executed by an External DO can update variables in the existing map set only if you pass the variable names to the program. However, from within the program executed by an External DO, the user can update field variables in the existing maps if they are reconversed or if another screen is conversed over them with an UPDATE.

The argument names passed in the originating program's DO statement need not match the corresponding parameter names in the external program's ENTRY statement. The following entry accepts MAP1, MAP2, FIELD1, and FIELD2 as M1, M2, F1, and F2:

```
DO PROG2(MAP1,MAP2,FIELD1,FIELD2)
...
ENTRY PROG2(M1,M2,F1,F2)
```

Once you pass the maps (MAP1 and MAP2), you can CONVERSE or CLEAR them and update variables (FIELD1 and FIELD2) within the external program (PROG2). The passed maps and variables need not be reinitialized with SCREEN or LET statements.

In order to enter variables into previously existing maps while executing an External DO, you must perform another CONVERSE. To enter data in multiple maps, CONVERSE UPDATE the last conversed map. Or CONVERSE UPDATE a map defined within the program executed by an external DO over the existing maps. In the following example, the fields in MAP1 may be updated from either CONVERSE:

```
ENTRY PROG1
.
.
.
SCREEN MAP1("A")
CONVERSE MAP1 WAIT
DO PROG2
.
.
.
ENTRY PROG2
SCREEN MAP2("B")
CONVERSE MAP2 UPDATE
```

## Parameter passing

External DO works like the internal DO in parameter passing. The following considerations apply:

- ◆ Data symbolic names are passed “by reference” only. Actual data are not passed “by value.”
- ◆ You must define all data names before using them on a DO statement. BIG, SMALL, KANJI, DBCS, TEXT, LET, SCREEN, FILE, TOTAL, ACCESS, INTERFACE, VIEW, SHOW, and OBTAIN can be used to define a variable.
- ◆ Data referenced by parameters are global to both the calling routine and the subroutine. Modified values are retained on return.
- ◆ A maximum of 255 parameters can be passed in one DO statement.
- ◆ Only passed parameters can be referenced outside the routine where they are defined.
- ◆ Reserved words, literals, expressions, or individual elements of an array (e.g., A(5,4)) cannot be passed as parameters. However, the entire array (e.g., A) can be passed.
- ◆ SCREEN, FILE, ACCESS, TOTAL, VIEW, and INTERFACE variables can be passed as parameters, but the individual fields within these are not available unless the individual fields are passed as parameters.
- ◆ Parameter passing can affect automatic mapping. See “[External DO programming guidelines](#)” on page 538 for details.

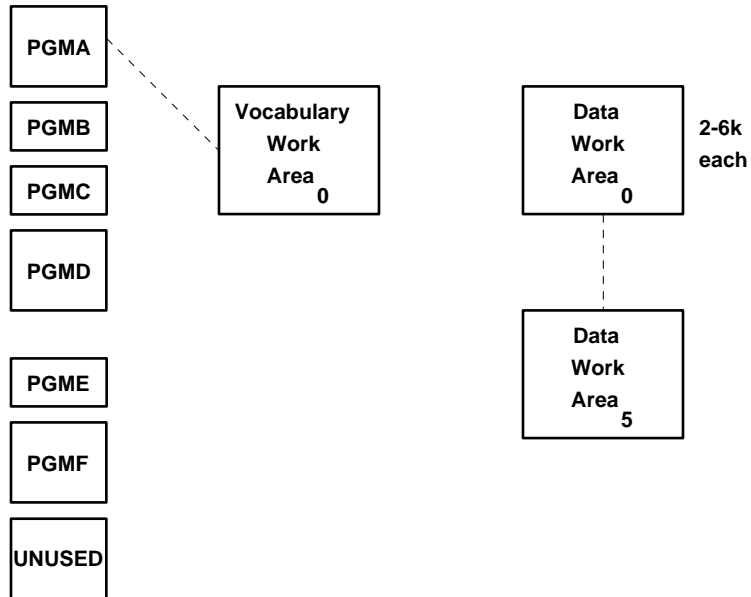
## Program architecture

The following figure illustrates the activity of a program that contains an external DO. When the PROGRAM statement is executed, MANTIS loads the program into a Local Program Chain.

When MANTIS encounters a DO statement, it locates the program in the Local Program Chain and allocates new data work areas. Because MANTIS is reentrant, the program involved by the DO statement is executed directly from the Local Program Chain.

When the external routine hits EXIT, its program and data work areas are released. The program is retained in the stack until it is needed again. Processing resumes with the statement following the DO in the previous level's program and data work areas.

#### Local Program Chain



A maximum of five DOLEVELs in addition to the zero level are allowed. The number of external programs executed at each level is unlimited. However, memory storage utilization increases because each program is loaded and kept on the program stack. The program stack holds the code for all programs that have been defined by execution of a PROGRAM statement. The program stack is only released when you issue a "QUIT", "NEW", or "CHAIN" in programming mode, or "STOP", "KILL" or "CHAIN" on a program running outside programming mode. Individual programs can be removed by RELEASE program-name.

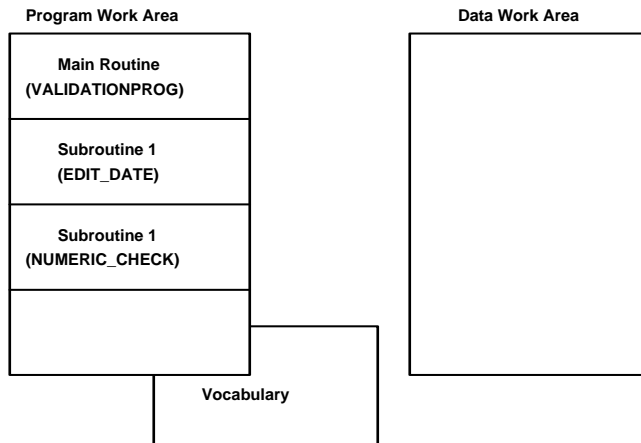
External DO maintains context for the task, along with all associated program storage, in main memory for as long as possible. In this way external DO reduces I/O to the MANTIS cluster.

## Internal DO vs. external DO vs. CHAIN

This section describes the use of external DO, internal DO and CHAIN. A basic understanding of the characteristics of each will help you apply the specific programming guidelines contained in the next section.

The following figure is an example of an internal DO. This is a main line routine bounded by an ENTRY and EXIT statement. The DO statements indicate subroutines that are physically resident with the calling routine.

The execution involves one program work area (up to 64K), one data work area (up to 64K) and one VWA (limit 2048 variables). Up to 255 parameters are passed, but all (up to 2048 variables) can be global. MANTIS stores the program as one entity.



```
ENTRY VALIDATIONPROG
```

```
.
.
.
```

```
DO EDIT_DATE (MAP , , , )
```

```
.
.
.
```

```
EXIT
```

```
ENTRY EDIT_DATE( , , , )  
.  
.  
.  
DO NUMERIC_CHECK( , , , )  
.  
.  
.  
EXIT  
ENTRY NUMERIC_CHECK( , , , )  
.  
.  
.  
DO MARK_FIELD( , , , )  
.  
.  
.  
EXIT  
ENTRY MARK_FIELD( , , , )  
.  
.  
.  
EXIT
```

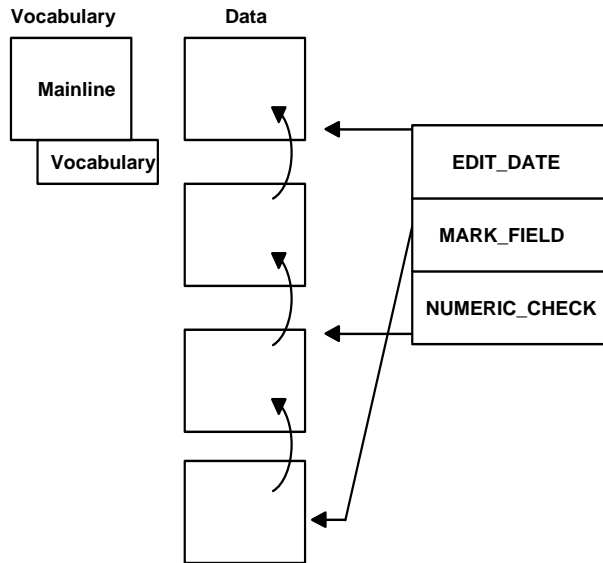
The following figure is an example of an external DO where the Shared Pool is not used. You must use ENTRY and EXIT statements around each physical program. (Dash lines indicate physical program boundaries.)

In this example, each of the four program areas can be up to 64K allowing more space for program logic. Each data work area can also be 64K, and each Vocabulary Work area can be up to 64K. All user words are local to their own program. Therefore, everything the subroutine uses from its caller must be defined on the ENTRY statement. The parameter list of user words is by reference only. No data is moved to an external data area. Up to 255 parameters can be passed; variables not passed are local to each routine.

Each time a DO is executed, a new data area for the subroutine is allocated and rebuilt.

External program code is executed directly from the Local Program Chain and is retained in the Local Program Chain when you EXIT. External program data area is released at EXIT. The PROGRAM statement does the actual external program loading onto the Local Program Chain.

### Local Program Chain



```
ENTRY VALIDATEPROG
DO EDIT_DATE(MAP,,, )
DO MARK_FIELD
EXIT
-----

ENTRY EDIT_DATE(,,, )
DO NUMERIC_CHECK(,,, )
EXIT
-----

ENTRY NUMERIC_CHECK(MAP,,, )
DO MARK_FIELD(,,, )
EXIT
-----

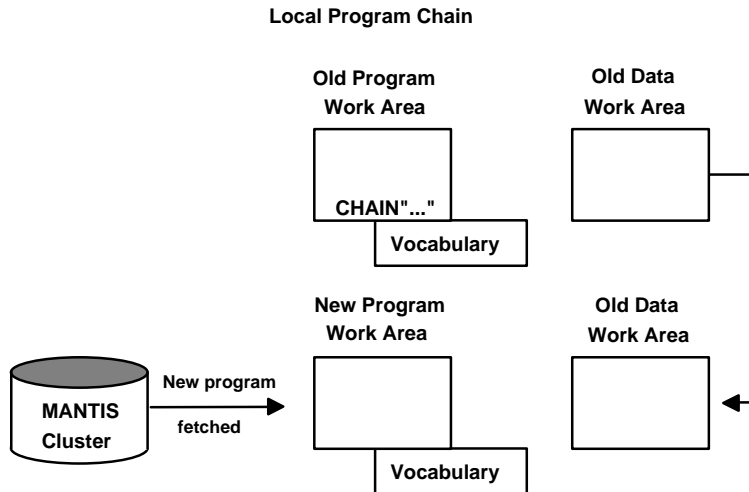
ENTRY MARK_FIELD(,,, )
EXIT
```

A program is not added to the Local Program Chain when the program already resides in the Shared Pool.

The following figure illustrates a program using the CHAIN statement. An ENTRY and EXIT statement is required around each physical program. (Dash lines indicate physical program boundaries.)

As in external DO, programs can be up to 64K. Each data work area and each program work area can be up to 64K. Each user word table can contain 2048 variables. Up to forty variables can be passed.

When MANTIS encounters a CHAIN statement, it fetches the program from the library or shared pool. A new data area is allocated and built. Parameters are copied from the old data work area into a new data work area. When new areas are established, the old program and data work areas are released.

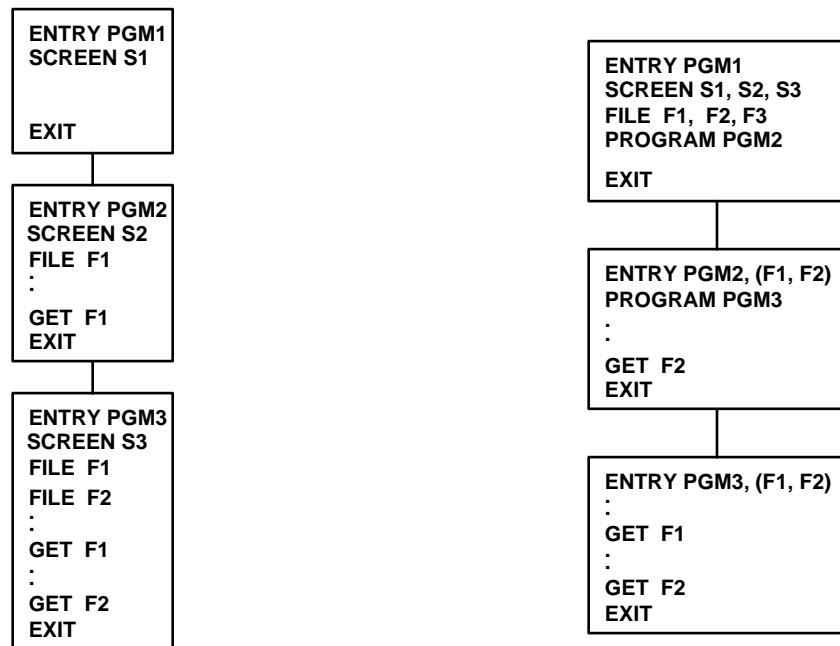


```
ENTRY OLD
    CAIN "NEW,A,B,C"
EXIT
-----
ENTRY NEW (X,Y,Z)
.
.
.
EXIT
```



Referring to the application on the left in the following figure as an example, let's examine how using different statements affects the number of I/O accesses against the directory. If you use CHAIN to link PGM1, PGM2 and PGM3, each time CHAIN is encountered MANTIS fetches the new program from the library. Complex variables (SCREEN and FILE) must be defined in each new data work area. If you use internal DO, you decrease the number of I/O accesses since everything you need to execute is loaded once, which saves fetching programs and screens.

External DO (the figure on the right) allows you to define in the main program variables that are used in all routines. After definition in the main program, they can be passed to subsequent routines. For example, files F1, F2 and F3 are defined in PGM1, causing one I/O access and can be passed to PGM2 and PGM3 with no I/O access required. The only I/O accesses required are those to fetch PGM2 and PGM3 the first time they are refreshed.



## External DO programming guidelines

Always use external DO with the following guidelines in mind.

### Program statement

Without the program in the shared pool, when the PROGRAM statement is executed, storage is allocated and library I/O occurs to load the program into the program stack. You may defer the execution of the PROGRAM statement as long as possible to avoid unnecessary storage acquisition and initial I/O on the MANTIS cluster. Do not group all PROGRAM statements at the start of the main program. The logic of the main program can preclude execution of some of the named programs, thereby wasting the storage that these PROGRAM statements acquire. The best method is to pair the PROGRAM statement and its corresponding DO. Note that encountering a PROGRAM statement the second time during program execution involves minimal overhead.

The following examples illustrate the suggested way to use external DO in your program:

### DON'T

```
10 PROGRAM PROG_1 ("PROGRAM_1",PASSWORD)
20 PROGRAM PROG_2 ("PROGRAM_2",PASSWORD)
30 PROGRAM PROG_3 ("PROGRAM_3",PASSWORD)

.
.
.
100 IF OPTION=1
110 .DO PROG_1
120 ELSE
130 .IF OPTION=2
140 ..DO PROG_2
150 .ELSE
160 ..DO PROG_3
170 .END
180 END
```

**DO**

```

100 IF OPTION=1
110 .PROGRAM PROG_1("PROGRAM_1",PASSWORD)
120 .DO PROG_1
130 .ELSE
140 .IF OPTION=2
150 ..PROGRAM PROG_2("PROGRAM_2",PASSWORD)
160 ..DO PROG_2
170 .ELSE
180 ..PROGRAM PROG_3("PROGRAM_3",PASSWORD)
190 ..DO PROG_3
200 .END
210 END

```

Group other entities such as screens or files with the PROGRAM statement if they are passed to that program. For example:

```

WHEN KEY="PF1"
. PROGRAM PROG_A("PROGRAMA",PASSWORD)
. SCREEN MAP_A("MAPA")
. FILE RECORD_A("FILEA",PASSWORD)
. DO PROG_A(MAP_A,RECORD_A)
END

```

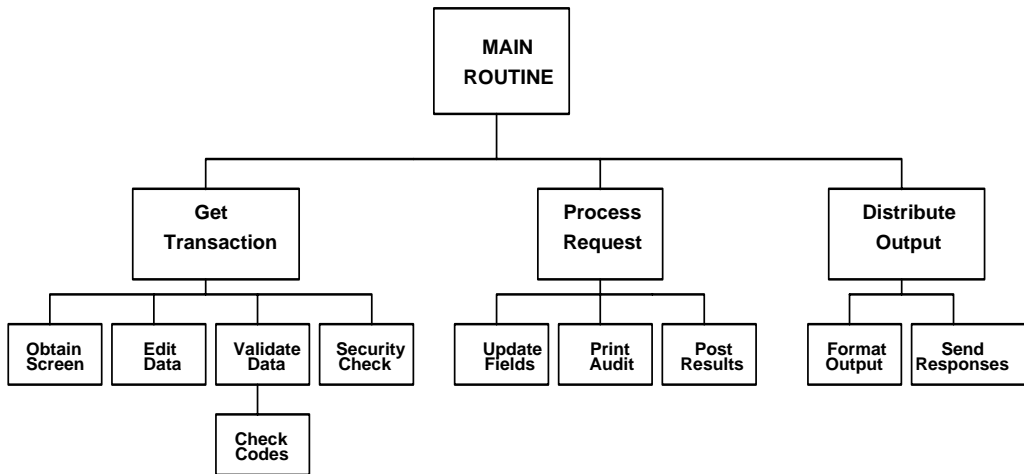
It is best to defer execution of statements that cause library I/O (SCREEN, FILE, ACCESS, TOTAL, VIEW, INTERFACE) if they are not certain to be needed. This is even more important with the PROGRAM statement because of the potential program size. When using HPO Bind, you may want to include statements like SCREEN or FILE where they can become HPO bound.

## Modularization

Many small externally done programs in the system cause overhead. Each program requires at least one I/O to the MANTIS cluster, plus extra processing and storage for each PROGRAM and DO statement. When you have many small routines to execute, it is advisable to group related routines into one program, then pass a parameter to the program, indicating the routine to be executed. Or you can pass control to the main routine in the program, and have it internally DO the subordinate routines.

Where the system has been designed via a structured methodology, multiple related modules can be combined into one program. For example, the following figure shows how you can combine 14 separate modules into three programs. In other words, do not make every module a separate program. Instead, mix internally and externally done routines.

Routines that are used throughout an application, or on a shopwide basis, can still be placed internally in a program and maintained in one place if the Component Engineering Facility is used. In maintaining the components, you can cut down on the overhead when you are making widespread changes. Refer to *MANTIS Program Design and Editing, OS/390, VSE/ESA*, P39-5013, for more information.



## Automatic mapping

In order for automatic mapping of variables to occur, you must define complex statements (SCREEN, FILE, and so on) at the same level, *or* pass them as parameters. Each subvariable is local to the routine in which it's defined. The same is true of keys on INSERT or GET statements.

## Examples

When defining complex variables at the same level, the fields in the record to be inserted are automatically mapped from the screen. The CONVERSE could be in either SUB1 or the calling routine with the same results.

```
30 SCREEN MAP("X"): |CONTAINS F1,F2,F3
40 FILE REC("Y","P"): |CONTAINS F1,F2,F3
50 DO SUB1(MAP,REC)
```

automatically maps F1, F2, and F3 between MAP and REC.

```
10 ENTRY SUB1(SC,FI)
20 .CONVERSE SC
30 .INSERT FI
40 EXIT
```

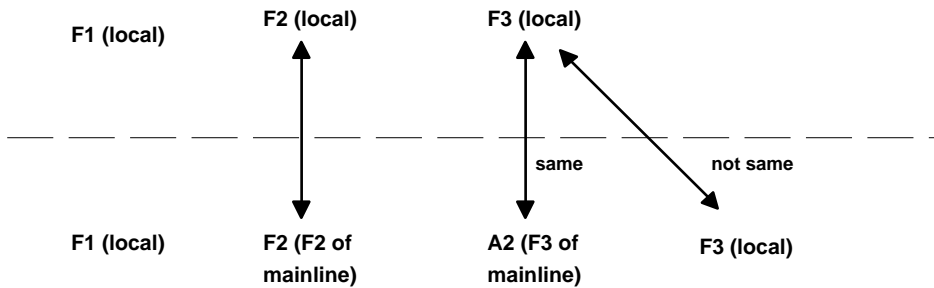
In another example, F1, F2 and F3 are defined in the main routine. On entry to the subroutine, SC, F2 and A2 are defined as arguments in the subroutine. When the FILE statement executes, MANTIS looks for F1 and doesn't find it, so it defines one. When it hits F2, it finds an F2 (because it appeared on the ENTRY statement) and connects to it. When F3 is hit, MANTIS doesn't find it, so it defines another F3 (not the one in the main routine). A2 in the subroutine is the local name for the calling routine's F3 and shares its data area.

The CONVERSE statement fills in all the variables in SC. The only variable automatically mapped in the CONVERSE and INSERT is F2.

Because F2 is defined as an argument on the ENTRY statement, it is automatically mapped between the file and the screen. Even though F3 is passed as a parameter, the name F3 is not known to the subroutine SUB1. Therefore, the FILE statement establishes a local variable F3 (as well as F1) in SUB1.

```
10 ENTRY MAINLINE
30 SCREEN MAP("X"): |CONTAINS F1,F2,F3
50 .DO SUB1(MAP,F2,F3)
10 ENTRY SUB1(SC,F2,A2)
20 FILE REC("Y","P"): |CONTAINS F1,F2,F3
30 CONVERSE SC
40 .INSERT FI
```

The diagram below shows what happens to the passed variables.



## Terminal I/O

If you place a SHOW statement in a lower level of your hierarchy, you can lose data. The first SHOW statement must be at the same level of the first I/O within your program. If a SHOW is in a lower level without a WAIT statement, the SHOW is deleted.

If a view is defined in a caller and passed as a parameter on an external DO, both the view and the field-name must be passed.

Also be advised that when you use a CONVERSE SET, CONVERSE WAIT, or a CONVERSE UPDATE with a screen defined at a lower level, when the program exits that level the whole map set is deleted.

## Entity definition

Avoid entity definition within (non-HPO-Bound) external programs because this causes recurring I/O's against the directory file and rebuilds the user word table. Declare all complex variables (SCREEN, FILE, TOTAL, ACCESS, and so on) as high up as possible in the hierarchy of external DOs. However, keep in mind the constraints for automatic mapping mentioned in "Automatic mapping" on page 541.

The data work area is released on EXIT and reacquired on each DO. If you have (non-HPO-Bound) complex statements in subordinate routines, they must be refetched from the library (causing I/O) and rebuilt (causing CPU cycles used) each time the program is done. Even if you cannot define these entities at the top level (DOLEVEL=0), the higher you can place these statements, the better.

However, you can use external DO to your advantage when you need to redefine variables. For example:

```
ENTRY MAIN
.TEXT FILE_NAME(33),FILE_PASSWORD
.UNTIL KEY="CANCEL"
..SHOW "ENTER FILE NAME AND PASSWORD TO CLEAR"
..OBTAIN FILE_NAME,FILE_PASSWORD
..DO SUB(FILE_NAME,FILE_PASSWORD))
,,SHOW FILE_NAME,"SUCCESSFULLY CLEARED"
.END
EXIT
ENTRY SUB(F,P)
.FILE REC(F,P)
.DELETE REC ALL
EXIT
```

If SUB is an internal routine, the symbolic name, REC, is defined on the first iteration. Subsequent iterations will not redefine REC, so only one file can be used. If SUB is an external routine, each execution starts with a new data work area, so REC will be defined on each iteration, making it possible to clear any number of files.

### **Frequency of calls**

The more often your application uses a subroutine, the greater advantage you have using DO over a CHAIN statement (with a CHAIN to return). This is because the calling routine context is maintained, and the called routine does not need to be fetched from the library each time.

However, if an external routine can be designed to do as much work as possible per call, performance is improved. For example, if a routine can be called once to validate all elements of an array, it is better than calling repetitively for each element to be validated. Very heavily used routines are best performed as internal routines.



## Debugging

Using external DO in programming mode allows you to debug interactively. Use the EXIT statement to move to the previous or calling routine. To get back to the subroutine after exiting, enter RUN and the line number of the DO statement. Use the “PROGRAM==>” or “EDIT—” heading to determine the program that you are in.

Use SHOW DOLEVEL to see what level is executing. You can modify and replace programs at any level. The revised version is executed in the next run. Note that if you don’t SAVE or REPLACE the copy in the workspace before exiting, your changes go away upon EXIT.

Remember that arithmetic and text variables, lists or arrays must correspond in datatype between the ENTRY-EXIT and the DO or CHAIN statements.

## Conversion

When deciding whether to convert existing applications using chaining to use external DO, remember that with external DO all variables must be passed. Any complex variables (SCREEN, FILE, or any variable that has other subvariables fetched from the library) must pass subvariables on ENTRY and DO.

Use external DO:

- ◆ When control must be automatically returned to where it was invoked.
- ◆ When a routine is common to many programs.
- ◆ When user table limits (2048 variables) are reached.
- ◆ When program size is exceeded.
- ◆ When data area size is exceeded.
- ◆ When excessive amounts of chaining occur between a number of small programs.

The internal DO is best used:

- ◆ When control must be returned to where invoked.
- ◆ When many data items must be shared.
- ◆ When a routine is used heavily in the program.
- ◆ When a routine is small in size.
- ◆ When a routine does not need to be called from other programs.

Continue to use CHAIN:

- ◆ When control can be returned to the beginning of a routine.
- ◆ When a return path is not necessary.
- ◆ When the return path is unlikely or infrequent.

Use components:

- ◆ When external DO is not required or desired.
- ◆ When the common code is not a logically complete unit (that is, when it is a program fragment).
- ◆ When program fragments are shared or common among many programs.
- ◆ When program fragments are shared and are likely to change.
- ◆ To include in-line code, internal subroutines, or related groups of internal subroutines.

## Program size

- ◆ The maximum program size is 64K. However, making every externally done program in the system close to the maximum program size tends to stress the system.
- ◆ It is a good idea to use moderation in the size of your programs. Try not to go to either extreme, that is, a few very large programs or a great number of small programs. Use the PROGFREE built-in function to determine program size.
- ◆ If you have many large programs occupying memory as part of the hierarchy of external DOs, the amount of storage needed to run MANTIS applications systems could increase. CICS MANTIS users may need to increase the size of the CICS Dynamic Storage Area.

## Programming techniques

Using external DO, you can use the same file statement to reference many file descriptions because a new FILE statement is created for each call. For example:

### Program 1

```
.  
.   
PROGRAM FILENEW( "PGM2" , "PASSPGM2" )  
.   
.   
.   
.   
FILE_ID=LIBNAMEX  
INSERT_LEVEL=INSERT_PASSWORD  
DO FILENEW( FILE_ID , INSERT_LEVEL )  
.   
.   
. 
```

### Program 2

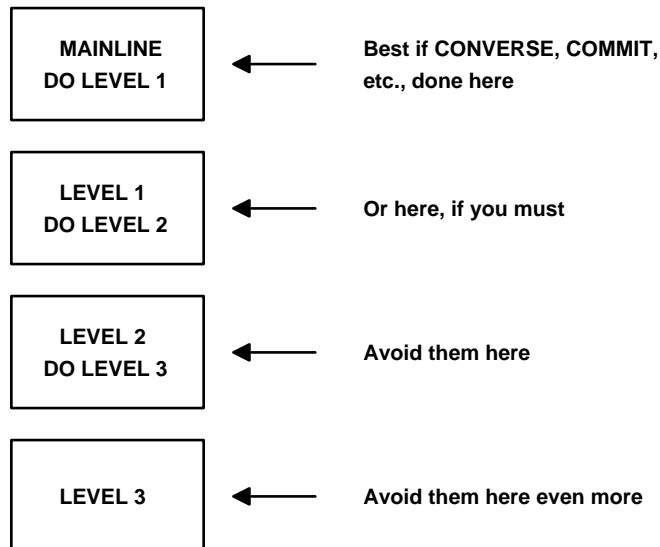
```
ENTRY PGM2( FILE_NAME , PSSWD )  
.   
.   
.   
FILE F( FILE__NAME , PSSWD )  
.   
.   
. 
```

You can change FILE\_ID and INSERT\_LEVEL in program 1 before any DO statement. MANTIS creates a new file statement for each call. This also works for other complex variable type (such as SCREEN, FILE, ACCESS,).

## Rollout/Rollin

In the CICS pseudoconversational environment, avoid coding any statement that causes a terminal I/O or context save in any externally done routine, such as SHOW, WAIT, PROMPT, PERFORM, CONVERSE and OBTAIN in an externally done routine. In these environments all program context from the currently executing routine, including all previous active levels back to the main routine, has to be rolled out and back in at terminal I/O. Because external DO can maintain a larger task context than internal DO, rolling out and in against this extra context can have a negative impact on performance.

It is better to handle all screen I/O in the main program, and handle all calculations, conversions and editing in externally done routines. This is not always possible. If external routines must contain such commands, try to place them in the higher level external routines rather than the lower level routines, as shown in the following figure:



## VSAM deadlocks

A VSAM deadlock is created when two transactions each wait for a resource held by the other transaction. For example:

- ◆ Transaction A holds record 1
- ◆ Transaction B holds record 2
- ◆ Transaction A requests record 2 and waits
- ◆ Transaction B requests record 1 and waits

## VSAM Files

When a record from a VSAM file is held, then the control interval that contains the record is held, therefore, all the records in the control interval are unavailable. For recoverable files in CICS, the control intervals are held for the duration of the logical unit of work.

In order to free the held control interval, a COMMIT or RESET instruction must be executed.

For nonrecoverable files, the control intervals are held for the duration of the execution of the commands INSERT, UPDATE, and DELETE.

The instructions that hold a control interval are the following:

- ◆ INSERT
- ◆ DELETE
- ◆ UPDATE
- ◆ ENQUEUE (with GET)

The commands that free a control interval by completing a logical unit of work are as follows:

- ◆ COMMIT
- ◆ OBTAIN\*
- ◆ RESET
- ◆ PROMPT\*
- ◆ CONVERSE\*
- ◆ WAIT\*
- ◆ SHOW (filling up the screen)

\* When COMMIT ON is in effect.



---

When the MANTIS transaction ends, all the control intervals are freed.

---

## Deadlocks on GET NEXT

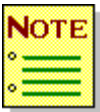
When you read a file using the GET NEXT statement with or without a key, a GET FIRST, GET LAST, or a GET PRIOR, MANTIS generates the following commands:

```
EXEC CICS START BROWSE  
EXEC CICS READNEXT
```

A START BROWSE holds a string for the VSAM file. An END BROWSE frees the string.

The MANTIS statements that generate an EXEC CICS END BROWSE are listed in the categories below:

- ◆ Statements that free all strings for a transaction:
  - CALL
  - COMMIT
  - CONVERSE
  - OBTAIN
  - PERFORM
  - PROMPT
  - RESET
  - SHOW (when the screen is full; forcing I/O)
  - WAIT
- ◆ Statements that free ONLY strings held by a previous GET:
  - DELETE
  - GET EQUAL with KEY
  - INSERT
  - UPDATE



---

If the GET NEXT reads to the end of the file, the string is freed by an END-BROWSE at that time.

---



## Rules for avoiding deadlocks

- ◆ Put the commands that hold control intervals as close as possible to the instructions that free them.
- ◆ Applications that hold several records at the same time must always hold them in the same order; for example, ascending key sequence.
- ◆ Programs that update more than one file must always update them in a predefined order. Each program must update the files in the same sequence as another program using the same files.
- ◆ The ENQUEUE resulting from a GET with ENQUEUE is released by any of the following statements:
  - COMMIT
  - CONVERSE
  - DELETE
  - DEQUEUE *file-name*
  - INSERT
  - OBTAIN
  - RESET
  - SHOW
  - UPDATE
  - WAIT
- ◆ Never code more than one ACCESS command for the same file in a program if local shared resources are being used.

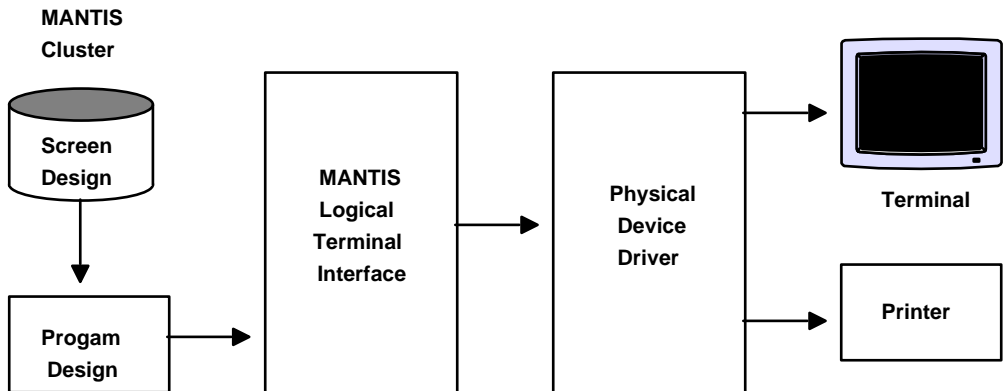
For additional information concerning deadlocks, refer to the *CICS/VS Application Programmers Reference Manual*.



# E

## Enhanced screen and program design

MANTIS has a logical interface between itself and the physical device environment, thereby removing IBM 3270 terminal dependence. The MANTIS Screen Design Facility enables you to design and save screens for use in your application. The MANTIS Program Design Facility allows you to build programs that send your screen designs to physical devices. The following figure illustrates the interaction among the MANTIS cluster, the MANTIS Logical Terminal Interface, and the physical devices:



MANTIS uses two steps to send a map (screen) to a terminal or printer.

1. Screen designs are sent to the MANTIS Logical Terminal Interface through programming commands (such as CONVERSE).
2. MANTIS communicates with the Physical Device Driver to send output to one or more physical devices (terminal or printer), as appropriate. The Physical Device Driver ignores specified features not supported by the physical device.

## Designing screens

The MANTIS Screen Design Facility supports a logical display of 255 rows by 255 columns which enables you to design and converse screens. Your physical screen acts as a moveable window on the logical display. By using PF keys, you can scroll around the logical display.

The following figure illustrates the design area used to create a screen. The bottom two lines of this screen are reserved for the message line, the row/column coordinates (window mode only), the command line, and the key simulation field. MANTIS also reserves the first three columns of the design area for the row scale line. You can remove this row scale line to make use of the full screen width, if appropriate.

```
.
.
.
.
+
.
.
.
.
1
.
.
.
.
+
.
.
.
.
2
.
.
#####          ##      ##
#####          #####
```

To create the screen, use the Create or Update a Screen option from the Screen Design Facility. Next, define attributes for each data field on the screen. As part of field definition, you can specify multiple vertical or horizontal occurrences of a particular field. Also, you can indicate whether a field will appear in a particular color or intensity. Extended edit characteristics allow you to specify edit values for a field which reduces edit code in your program. You can designate other attributes on a map level.

The following table shows the MANTIS Screen Design field attributes (described in more detail in *MANTIS Facilities, OS/390, VSE/ESA, P39-5001*):

Field attribute	Description
FIELD NAME	The name used by MANTIS to refer to the field.
LENGTH	The length of the field. If the length of the field has been specified using hash characters (#) on the screen design, that length will be displayed here. Some designers enter the field length here and not on the screen design, again highlighting the flexibility of MANTIS.
ROW/COLUMN	MANTIS supplies the row and column position of the field on the screen.
DATA TYPE	Data entered in the field must be text, numeric, or DBCS.
INTENSITY	The field will be bright (highlighted), display normally, or be hidden (not displayed).
CURSOR	Positions the cursor in the field when screen is displayed.
PROTECTED	If PROTECTED=Yes then the field is read-only; if No, then the field can be used for display and data entry.
AUTOSKIP	Automatically skips the cursor to the next unprotected field when user fills the current field.
UPPERCASE	If UPPERCASE=Yes, data-entry text fields will be translated to forced uppercase. If No, the field will accept upper and lowercase characters.

Field attribute	Description
BLINKING	Data in the field will blink when displayed.
REVERSE VIDEO	If Yes, the field will be displayed as dark characters on a light background or vice versa.
HIGHLIGHT	Field will be highlighted upon display.
COLOR	Specifies the color of the display if the terminal supports color.
MODIFIED TAG	Indicates whether the field has been modified.
DETECTABLE	If Yes, the field is pen-detectable (if terminal supports pen-detectable capabilities).
BOX	Allows you to draw a box around a field if the terminal supports this feature.
SO/SI	DBCS support terminals only. Allows DBCS and EBCDIC characters to reside concurrently in a text variable.
REPEATS*	Indicates how many times a field will occur horizontally and vertically on the screen and the space between occurrences. Repeat specifications can also be set using the Update Repeat Specifications option on the Screen Design Facility menu.
EXTENDED EDIT	If EXTENDED EDIT=Yes, the field has additional attributes such as default values, range checking.

\* You can also specify repeats in the Update Repeat Specifications option in the Screen Design Facility.

The following table shows the MANTIS Screen Design map attributes (described in more detail in *MANTIS Facilities, OS/390, VSE/ESA, P39-5001*):

Map attribute	Description
MAP DOMAIN	Shows the maximum row and column coordinates for the current screen domain.
BLANK FILL CHARACTER	Specifies the character you want to use for blank fills.
SOUND ALARM	Indicates whether or not you want MANTIS to sound an alarm each time you converse the screen.
PROTECT BOTTOM LINE	Indicates whether you want MANTIS to protect the bottom line of the screen (Command Line and Key Simulation Field).
MASK CHARACTER	Specifies the character you want to use as the mask character to identify fields.
FULL DISPLAY	Indicates whether or not you want MANTIS to expand the screen size to the dimensions of the current terminal, including the bottom two lines of the screen.
OPAQUE MAP	Indicates whether or not a screen (map) will be opaque (rather than transparent) when it is conversed.
KEEP MAP MODIFIED	Prevents MANTIS from clearing the modified data tags of a specified screen.
SEND ALL FIELDS	Indicates whether all fields or just nonheading (data) screens are sent to the terminal.
RESET	Returns all field-level attributes to the original specifications made in Screen Design.

Special considerations for field attributes:

- ◆ You can suppress the last two lines on a displayed screen and extend default vertical repeats by specifying the FULL DISPLAY attribute in Library Functions. If you specify FULL DISPLAY, error messages will not display at the bottom of your screen.
- ◆ You can protect only the command line and key simulation field from input with the Protect Bottom Line attribute in Library Functions. This attribute has no effect if you specify the Full Display attribute. (Note that you cannot use the KILL mechanism to terminate a looping application program if you specify either of these attributes. We recommend that you not use these attributes just to inhibit use of the KILL word.) With the Protect Bottom Line attribute in effect, the user is unable to enter window mode. Therefore, do not specify this attribute for a screen that is larger than the physical display.
- ◆ Changing the blank-fill character from a vertical bar to another character permits you to use the standard blank-fill character (the vertical bar) as an output character.

MANTIS saves the blank-fill character with the screen design and returns it when you fetch the screen.



## Building a map set in your program

Individual maps can be added to the logical display to form a map set. Maps within a map set are displayed according to their entry sequence into the map set as specified in programming statements and overlay each other. The top map in a map set is known as the active map. Every field on this map that falls within the current window is displayed. All other maps in the map set are known as passive maps.

The position of a map within the map set is controlled by the first set of parameters on the CONVERSE statement.

---

```
CONVERSE screen - name [ [(row1,col1)] [ WAIT
                           SET
                           UPDATE ] [ WINDOW
                                     DISPLAY ] [(row1,col1)]
                           RELEASE ]
```

---

The WAIT, SET, and UPDATE options on the CONVERSE statement determine whether a physical I/O occurs (whether the map is displayed) and whether the displayed input fields on the passive maps are protected or unprotected. A CONVERSE statement overrides an ATTRIBUTE(PROTECTED) statement in your program, which overrides a protected field attribute specified in Screen Design.

Unless designed as opaque, all maps are translucent, meaning that, all fields not completely overlaid by the active map display through the active map. When conversing maps, the following rules govern whether you can update partially displayed fields on a passive map:

- ◆ Fields on a passive map that are partially displayed because they are overlaid by the active map cannot be updated.
- ◆ Fields that are partially displayed because they overlap the physical screen boundary can be updated.

The CONVERSE statement with the WAIT, SET, and UPDATE options is described as follows.

---

<b>CONVERSE</b> <i>screen - name</i>	[[ <i>row1,col1</i> ]]	[ <b>WAIT</b>	[[ <b>WINDOW</b>	[[ <i>row2,col2</i> ]]
		<b>SET</b>		
		<b>UPDATE</b>		
		<b>RELEASE</b>		

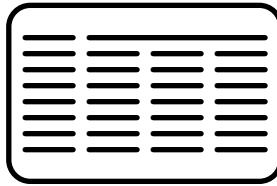
---

- ◆ CONVERSE *screen-name* without a WAIT, SET, or UPDATE creates a new map set. This map set contains only the named screen and causes a physical I/O to occur, thereby displaying the map on your terminal. The row/column coordinates (the default is row 1, column 1) specify the row and column positions for a screen.
- ◆ CONVERSE *screen-name* WAIT adds the named map to the current map set but does not cause a physical I/O to occur. The named map is not displayed.
- ◆ CONVERSE *screen-name* SET adds the named map to the current map set and causes a physical I/O to occur. This is the active map in the map set, that is, it appears on top of the other maps. This means overlapping fields from the active map have display precedence.
- ◆ CONVERSE *screen-name* UPDATE acts the same as CONVERSE *screen-name* SET, except all data entry fields that are fully displayed from underlying maps are unprotected. Unprotected fields that extend beyond the boundaries of the physical screen do not need to be completely displayed to be updated.
- ◆ CONVERSE *screen-name* RELEASE removes the specified map from the map set. If a CONVERSE map RELEASE is issued for a specified map that is not in the map set, MANTIS issues a fault.

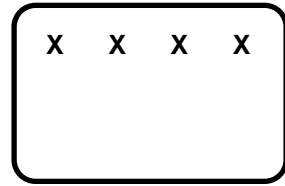
The final map display may consist of a single map or a composite set of maps positioned dynamically within the logical display. If the map display is larger than the physical screen, the operator can treat the physical screen as a window, moving it around the logical display by using window mode PF keys.

## The CONVERSE statement and mapping examples

The following screen designs show how one screen overlays the other:

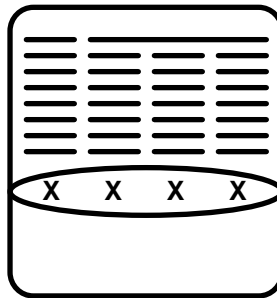


**Report Screen**



**Division Subtotals**

When you converse the maps into one display, you obtain the following results:



**Report Screen**

**Division Subtotals**

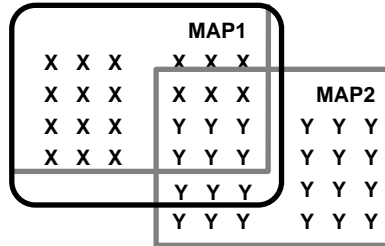
The following maps and program illustrate options of the CONVERSE statement and how active and passive maps are affected:

MAP1					
X	X	X	X	X	X
X	X	X	X	X	X
X	X	X	X	X	X
X	X	X	X	X	X

MAP2					
Y	Y	Y	Y	Y	Y
Y	Y	Y	Y	Y	Y
Y	Y	Y	Y	Y	Y
Y	Y	Y	Y	Y	Y

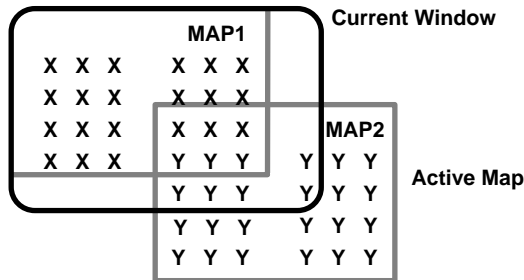
```
10 ENTRY CLIENT_ENTRY
20 .SCREEN MAP1( "NEW_CLIENT" )
30 .SCREEN MAP2( "PAGE_2" )
.
.
.
70 .CONVERSE MAP1
.
.
.
120 ...IF FIELD="Y"
160 ....DO EXTRA_INFO
.
.
.
300 ENTRY EXTRA_INFO
310 .UNTIL MAP2="CANCEL"OR MESSAGE=" "
320 ..CONVERSE MAP2(10,45)SET
.
.
.
400 .IF MAP2="ENTER"
410 ..INSERT REC
420 ..CLEAR
```

The CONVERSE statement in line 320 produces the following result. Note the fully displayed fields on MAP1 are protected because of the SET parameter used in the CONVERSE statement. Also, the last field on MAP1 is overlaid by the first field on MAP2.

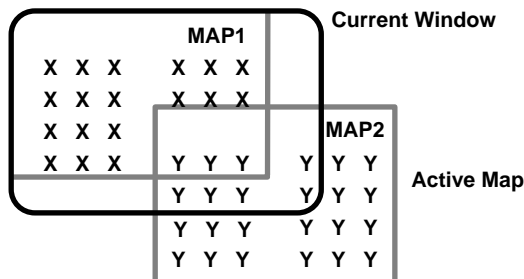


Below is a variation on the CONVERSE statement in line 320. This variation uses the UPDATE option instead of the SET option. Note that the fully displayed input fields on MAP1 are unprotected.

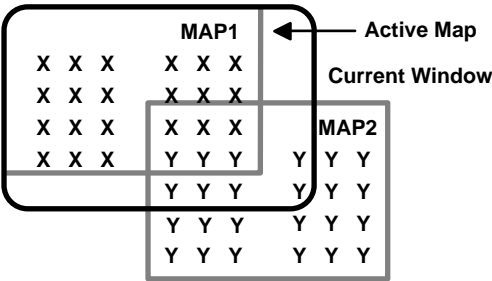
**CONVERSE MAP2(10,45) UPDATE**



You can specify an opaque map in screen design. The active opaque map completely overlays the parts of the passive map it covers, as shown in the following example:



Executing a CONVERSE MAP1 UPDATE again produces the following result:



As you are building your map sets, keep the following attributes in mind. You can set these attributes with the **ATTRIBUTE** statement (see “**ATTRIBUTE (Statement)**” on page 102 for more information), and you can set them in Screen Design (refer to *MANTIS Facilities, OS/390, VSE/ESA*, P39-5001, for more information).

**KEEP MAP MODIFIED**  
**RESET MAP MODIFIED**

**Description** Prevents MANTIS from clearing the modified data tags of a specified screen.

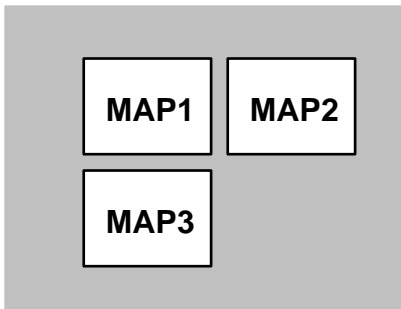
**Format** KEEP MAP MODIFIED or RESET MAP MODIFIED

**Considerations**

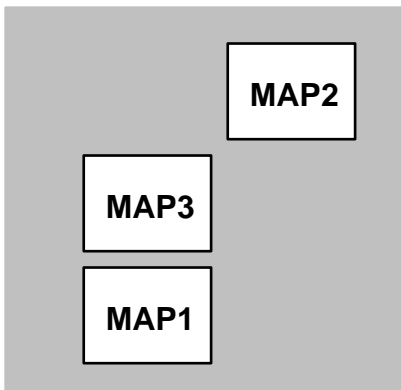
- ◆ Specify KMM to prevent MANTIS from clearing modified data tags or (MDT's) of the specified screen or MODIFIED (*map,field*) function. Ordinarily, MANTIS clears MDT's of all maps in the map set for a CONVERSE UPDATE, or for the active map in the case of a CONVERSE SET. If the MDT's are cleared, a previously modified map returns FALSE for the MODIFIED function. If a map has attribute “KMM”, then once modified, the MODIFIED function always returns TRUE. For an explanation of maps and map sets, see the CONVERSE statement.
- ◆ Specify Reset Map Modified (RMM) to turn off KMM and restore ordinary functionality.
- ◆ Use KMM if you will converse additional screens in the map set (e.g., pop-ups) and want to retain the MODIFIED setting for additional validation after the CONVERSE of the additional screen.

## Multiple images of a single screen design

A screen can only be conversed once in a map set. If you converse a screen a second time and specify row and column coordinates different from those in the first CONVERSE statement, the screen moves to the new location within the map set. In the following example, the CONVERSE statement in line 210 moves MAP1 from its original position (1,1) to its new position (60,1):



```
110 CLEAR
120 CONVERSE MAP(1,80)
130 CONVERSE MAP WAIT
140 CONVERSE MAP(30,1) SET
.
```



```
210 CONVERSE MAP1(60,1) WAIT
```

If you want one screen to appear multiple times within one logical display, you must define a new SCREEN symbolic variable for it each time you want it to appear. In the following example, MAP1 and MAP2 reference the same screen design entity:

```
10 SCREEN MAP1("NAME"),MAP2("NAME")
.
.
.
100 CONVERSE MAP1 WAIT
110 CONVERSE MAP2(45,1)WAIT
120 CONVERSE MAP3(15,1)SET WINDOW(15,1)
```

Each field on a screen has a unique name and value. If you are using the same screen design more than once in your logical display, assign multiple values to a field that appears in each screen occurrence. To do so, you need to prefix each successive occurrence of the screen as follows:

```
100 SCREEN MAP1("NAME",PREFIX)
```

In the previous example, MAP1 is the prefix that needs to be changed for each occurrence of the screen.



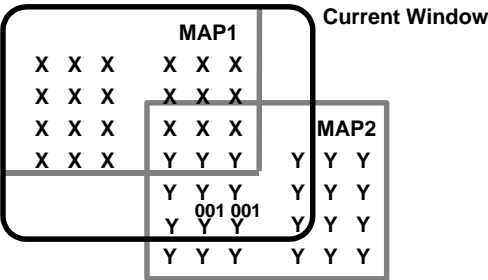
# Windowing

The CONVERSE statement can be extended to determine the physical display position of the map set. Because the logical display is 255 columns wide and 255 rows long, your map or map set may overrun the physical screen boundaries. By using the last parameters of the CONVERSE statement, you can specify whether window mode will be activated and where the physical display will be located on the map set:

```
CONVERSE screen - name[(row1,col1)] [ WAIT
SET
UPDATE
RELEASE ] [ WINDOW [(row2,col2)]
DISPLAY ]
```

CONVERSE *screen-name* WINDOW(*row,col*) enables window mode when the map is displayed, with the upper left corner of the window positioned at the specified row and column. This is known as program-initiated window mode. A window mode message, listing PF key settings, is displayed on the message line of the screen if no other message is displayed (e.g., an error message). The row and column coordinates of the upper left corner of the screen are displayed above the key simulation field. Sample displays are shown below. Note the fully displayed input fields on MAP1 are protected with the SET option and active with the UPDATE option.

```
CONVERSE MAP1 WAIT
CONVERSE MAP2(10,45)SET WINDOW
CONVERSE MAP2(10,45)UPDATE WINDOW
```



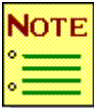
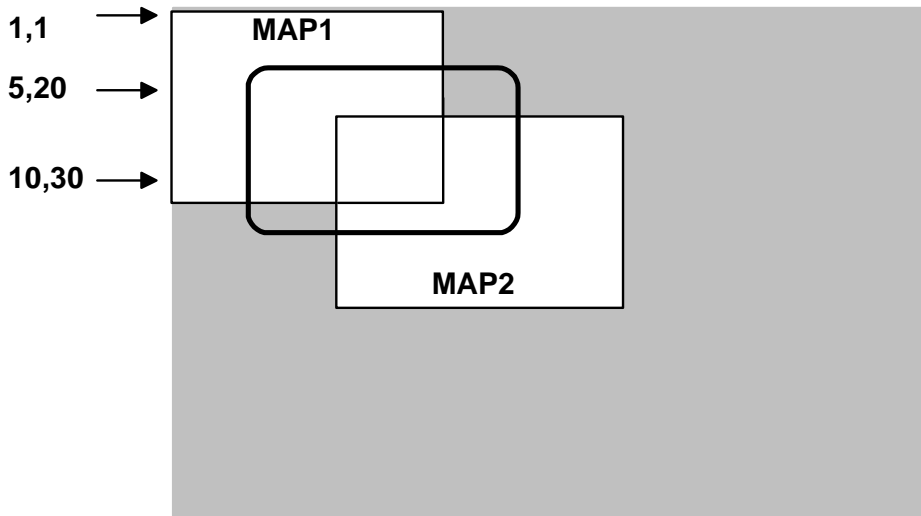
```
CONVERSE MAP2(10,45)UPDATE WINDOW
```

CONVERSE *screen-name* DISPLAY(*row,col*) allows you to position the physical screen at the supplied location without initiating window mode. However, *user-initiated window mode* can be activated by entering WINDOW (W) in the key simulation field of the conversed screen. *Automatic windowing mode* can be activated by entering AW in the key simulation field of the conversed screen. If you use AW mode and press PF12, the display is scrolled so that the cursor position is moved to position 1,1. Using this option, you do not have to get into, and back out of, windowing mode to move the display when the defined screen is larger than the physical display of your terminal.

---

### 210 CONVERSE MAP1(60,1)WAIT

---



---

The characters AW may be changed to other characters by your System Administrator.

---

## Window mode

In window mode, each time you press ENTER or a PF key, any valid updated fields on the screen are updated in the corresponding program variables. A CLEAR, PA1, or PA2 key affects only the fields entered on the last terminal I/O. Out of window mode, variables are not updated by a PA or CLEAR key.

The CANCEL key is defined by the Master User. When the CANCEL key is set to PA1, PA2, PA3, or CLEAR, and one of these keys is pressed, no data is moved from the screen's input fields. When the CANCEL key is set to something other than PA1, PA2, PA3, or CLEAR and a PF key or ENTER is pressed, data is moved to the screen's input fields (even if that key is designated as CANCEL).

All windowing occurs while the MANTIS program executes a single CONVERSE statement. You can reposition the physical screen by:

- ◆ Overtyping row and column values with new values and pressing ENTER. (e.g., enter 20 40 to scroll to row 20, column 40).
- ◆ Overtyping the row and column values with displacement values (+ or - in the first character position and the displacement value in the following character positions). For example, enter +10 -20 to scroll down 10 rows and left 20 columns.

In window mode, all PF keys are controlled by MANTIS and are not passed to the application. These settings are documented in *MANTIS Facilities, OS/390, VSE/ESA*, P39-5001. You can modify window mode PF key scrolling amounts within your program by using the SCROLL statement (e.g., SCROLL 30,30). The terminal operator can alter PF key scrolling amounts in window mode by overtyping the row and column values with incremental values (*i* in the first position and the incremental value in the following position). For example, enter i20 i80 to scroll down 20 rows (PF8) or up 20 rows (PF7) and right 80 columns (PF11) or left 80 columns (PF10).

The application program can initiate window mode (using the WINDOW option on the CONVERSE statement), but cannot restrict the positioning of the window by the operator. Therefore, the operator can override the PF key scroll settings specified in the program. The program can choose meaningful SCROLL increments for the initial scroll values.

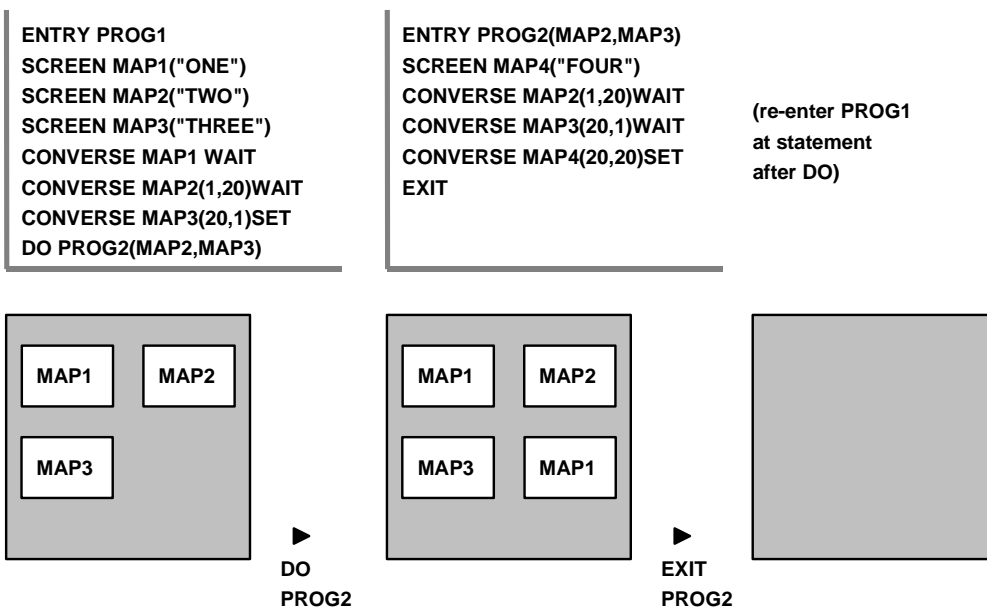
The application program can also use the DISPLAY (*row, column*) parameters to position the physical screen somewhere other than row 1, column 1 (upper left corner) of the logical display. DISPLAY does not initiate window mode, but the user may initiate window mode on the displayed map by entering WINDOW (W) in the lower right corner of the screen.

Note that input fields can be updated in window mode. In window mode, updated fields must fulfill extended edit requirements, if any, and are updated in the MANTIS program's variables when a scrolling PF key is pressed.

## Terminating window mode

Pressing PF6/18 or PA2 terminates window mode and returns control to the application program. Pressing PF6/18 causes MANTIS to return "ENTER" in KEY and screen variable name. In window mode, entering a value in the key simulation field and pressing PF6/18 returns control to your program and updates the screen variable name in your program. PA2 returns "CANCEL" to KEY and screen variable name. (PA2 is the default for CANCEL, but the Master User may assign another setting for CANCEL.)

The PF9/21 setting terminates window mode *without* returning control to the application program, thereby making window-mode PF key settings inactive. If you press a PF key when not in window mode, control returns to the application program, and the program receives the pressed key designation in KEY and screen variable name. The following figure shows the effects of terminating window mode using various keys:



To restrict the user from entering window mode and scrolling around a screen, you can specify the Full Display attribute or Protect Bottom Line attribute on the Library Functions option. The user is then unable to enter data in the key simulation field.

## Clearing a map

A conversed map remains in the map set until one of the following conditions occurs:

- ◆ The map set is cleared with a CLEAR statement.
- ◆ The map set is reset by a CONVERSE statement without a WAIT, SET, or UPDATE option.
- ◆ The map is removed with a CONVERSE *map* RELEASE.
- ◆ An external routine is exited where a map defined at that level is in the map set. See example 2 in “[Clearing a map set](#)” on page 575.

A CLEAR statement without a map name clears all conversed maps from the map set and sets the KEY function to CLEAR. CLEAR with a map name clears data from the data fields of the named map and sets the key value of the named map to null. It does not remove the named map from your map set.

CLEAR or CONVERSE (without a WAIT, SET, or UPDATE) clears the logical display of all existing maps. If you do not want to keep track of the first map set building CONVERSE statement, use a CLEAR statement prior to any CONVERSE. For example:

```
100 WHILE KEY<>"CANCEL"
110 .CLEAR
120 .WHEN NAME="Y"
130 ..CONVERSE MAP_NAME WAIT
140 .WHEN OPTION="Y"
150 ..CONVERSE MAP_OPTION WAIT
160 .WHEN REFERENCE="Y"
170 ..CONVERSE MAP_REFERENCE WAIT
180 .END
190 .CONVERSE MAP_SELECT SET
200 END
```

The CONVERSE statements (lines 130, 150, and 170) require no considerations if they are the first statements executed. CLEAR (line 110) clears any previous map set so these conversed maps do not overlay an existing map. The CONVERSE statement in line 190 sends the map to the logical display.

## Clearing a map set

A map set built within your program remains in the logical display until you CLEAR it. A map defined in an externally done program and not cleared before returning to the originating program causes the entire map set to be cleared. The following examples show how this rule works.

The examples also show how map names are passed to the external program by naming them as arguments of the DO statement in the originating program and as parameters of the ENTRY statement of the external program. The DO statement in PROG1 passes MAP2 and MAP3 as arguments to PROG2. PROG2 also uses MAP4, which is not defined in PROG1.

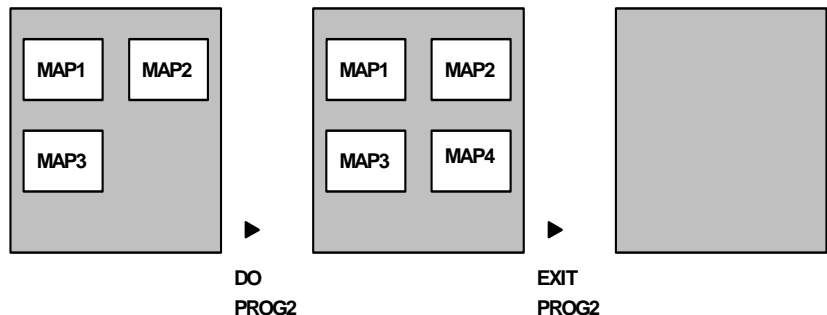
In Example 1, MAP4 is not cleared before exiting PROG2. Because MAP4 remains in the map set and is not defined in PROG1, the entire map set is cleared when returning to PROG1.

### //Example 1

```
ENTRY PROG1
SCREEN MAP1("ONE")
SCREEN MAP2("TWO")
SCREEN MAP3("THREE")
CONVERSE MAP1 WAIT
CONVERSE MAP2(1,20)WAIT
CONVERSE MAP3(20,1)SET
DO PROG2(MAP2,MAP3)
```

```
ENTRY PROG2(MAP2,MAP3)
SCREEN MAP4("FOUR")
CONVERSE MAP2(1,20)WAIT
CONVERSE MAP3(20,1)WAIT
CONVERSE MAP4(20,20)SET
EXIT
```

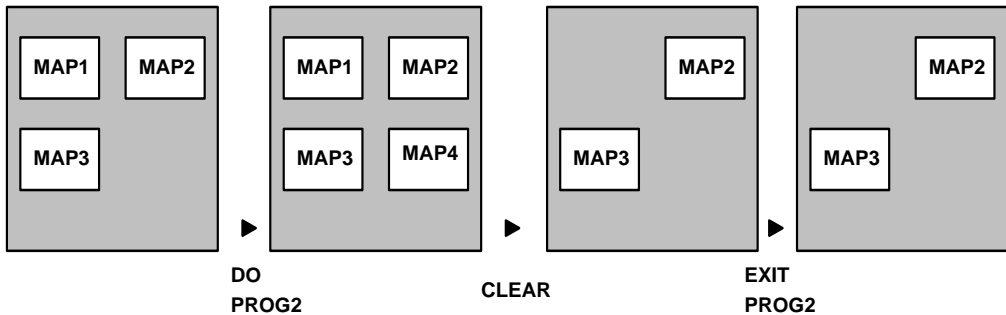
(re-enter PROG1  
at statement  
after DO)



In Example 2, PROG2 converses MAP4 and then clears it from the map set. Therefore, the map set, which now contains only maps already defined in PROG1, remains when returning to PROG1.

### Example 2

ENTRY PROG1	ENTRY PROG2(MAP2,MAP3)	( <i>reenter</i>
SCREEN MAP1("ONE")	SCREEN MAP4("FOUR")	<i>prog1 at</i>
SCREEN MAP2("TWO")	CONVERSE MAP2(1,20) WAIT	<i>statement</i>
SCREEN MAP3("THREE")	CONVERSE MAP3(20,1) WAIT	<i>after DO)</i>
CONVERSE MAP1 WAIT	CONVERSE MAP4(20,20) SET	
CONVERSE MAP2(1,20) WAIT CLEAR		
CONVERSE MAP3(20,1) SET	CONVERSE MAP2(1,20) WAIT	
DO PROG2(MAP2,MAP3)	CONVERSE MAP3(20,1) SET	
	EXIT	

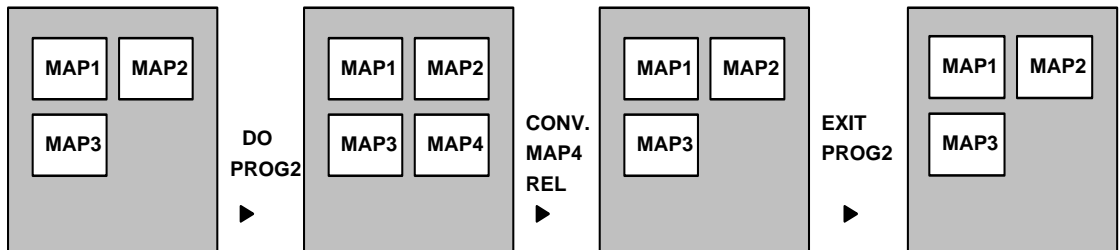




In example 3, the statement, CONVERSE MAP4 RELEASE, at the end of the program returns the map set to its original position.

### Example 3

ENTRY PROG1	ENTRY PROG2(MAP2,MAP3)	<i>(reenter PROG1</i>
SCREEN MAP1("ONE")	SCREEN MAP4("FOUR")	<i>at statement</i>
SCREEN MAP2("TWO")	CONVERSE MAP2(1,20)WAIT	<i>after DO)</i>
SCREEN MAP3("THREE")	CONVERSE MAP3(20,1)WAIT	
CONVERSE MAP1 WAIT	CONVERSE MAP4(20,20)SET	
CONVERSE MAP2(1,20)WAIT	CONVERSE MAP4 RELEASE	
CONVERSE MAP3(20,1)SET	EXIT	
DO PROG2(MAP2,MAP3)		



## Advanced editing

MANTIS can perform certain data editing checks (e.g., numeric field masks and range checks). Numeric field masks provide the following formatting capabilities:

- ◆ Float character
- ◆ Fill character(s)
- ◆ Floating + or - sign
- ◆ Trailing CR/DB/DR sign
- ◆ Zero filling

MANTIS performs extended editing and numeric field mask editing before returning control to the application program. MANTIS highlights any fields in error and displays a message for the first error. If MANTIS detects an error outside the physical screen (e.g., not currently displayed on the screen), MANTIS will automatically initiate window mode and will ask the user to locate and correct the field in error.

For more information on data editing, refer to *MANTIS Facilities, OS/390, VSE/ESA*, P39-5001.

# F

## Mixed-data support

Mixed-data is a data stream composed of any combination of SBCS (Single Byte Character Set), such as EBCDIC and DBCS (Double Byte Character Set). DBCS characters require 2-bytes (16 bits) to represent the character, as opposed to 1-byte (8 bits) for an SBCS character. DBCS characters are used to represent ideographic characters such as those in Kanji, Korean, or Chinese. You must set MIXMODE ON in order to have your program use mixed strings in TEXT variables. You should have MIXMODE OFF if you don't need it, both for efficiency's sake and for proper text operations when working with arbitrary binary values in TEXT variables (for example, MARK variables, Total refer, and CHR function).

The following conventions are used throughout this appendix to describe DBCS strings.

DBCS characters in TEXT type are separated from SBCS (Single Byte Character Set) characters by shift codes. In the following examples, a Shift-out of EBCDIC is noted by the < symbol, and a Shift-in to EBCDIC is noted by the > symbol, as shown below.

```
A="abc< 1 2 >de< 3 >fg"
```

In this example, the letters a, b, c, d, e, f, and g represent EBCDIC characters. The bold numerals represent DBCS strings.

In memory, SO and SI each occupy a single byte. On a terminal, they are normally displayed as a blank. The values for SO/SI codes are hardware dependent.

Mixed-data supported MANTIS runs on IBM55xx series terminals or other DBCS supported IBM55xx compatible terminals. Mixed-data is supported on all TP Monitors in the CICS and IMS environments.

## Using mixed-data in your program

Mixed-data is allowed in a MANTIS text variable or G-type (G“...”) literal. For example:

```
00010 TEXT ADDR(60)
00020 ADDR=G"< 1 2 3 4 5 >2-4-5 MORI< 6 7 >"
```

Shift codes are required only when a TEXT variable or a G-type literal (G“...”) contains some DBCS characters. A KANJI variable, DBCS variable, or K-type literal (K“...”) contains only DBCS characters, so no shift code is required.

A DBCS substring must contain an even number of bytes. The SO/SI must be balanced in a field when they are displayed. When mixed-data is entered from a terminal, MANTIS automatically balances the SO/SI pairs in the data stream. Zero length is valid for DBCS substring. For example:

```
00020 ADDR="AB<>CD"
```

where <> is called an empty DBCS. Shift-out (<) and Shift-in (>) are usually invisible when displayed or printed.

## Using mixed-data in screen design

Mixed-data support has been added to Screen Design in the form of the SO/SI attribute. This attribute is only valid with the text datatype. You can create SO/SI pairs for all screen design fields which possess the attribute.

The following screen illustration shows where the SO/SI attribute can be specified. As with many attributes, SO/SI can also be affected by the **ATTRIBUTE** statement. (See “**ATTRIBUTE (Statement)**” on page 102 for more information on this statement.)

B U R Y'S																							
CUSTOMER REPORT																							
CUSTOMER				CUSTOMER				BRANCH				CREDIT											
NUMBER				NAME				NUMBER				RATING											
												LIMIT											
#####	#####							####	##			\$#####											
*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
*	FIELD NAME : cust_number															:	LENGTH :	6	:	*			
*																ROW/COLUMN :	7	3	:	*			
*	DATATYPE		:	TXT		:	INTENSITY		:	NOR		:	CURSOR		:	N	:	*					
*	PROTECTED		:	N		:	AUTO SKIP		:	Y		:	UPPERCASE		:	N	:	*					
*	BLINKING		:	N		:	REVERSE VIDEO		:	N		:	HIGHLIGHT		:	N	:	*					
*	COLOR		:	NO		:	MODIFIED TAG		:	N		:	DETECTABLE		:	N	:	*					
*	BOX		:	L.	N	U.	N	O.	N	R.	N	:	SO/SI		:	N	:	*					
*	REPEATS		:	V.	14	1	H.						EXTENDED EDIT		:	N	:	*					
#####	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	###

OVERTYPE ATTRIBUTE SETTINGS AND HIT ENTER

## Heading fields

As the example below shows, you can use mixed-data and DBCS in screen design to specify mixed-data heading fields.

```

                                BARRY's
                                <%1%2%3>|REPORT

Customer| ||| |Customer| ||| |<%1%2%3>| ||<%1%2%3%4>|Credit
<%1%2>| ||| |<%%%%>| |||<%1%2%3%4>|Rating| ||| |Limit
#####  #####                                ##          $#####
```

## Screen design output and input and SO/SI pairs

MANTIS maintains integrity of the SO/SI pair in TEXT variables by automatically adding a SO or SI to complete a pair where one member may have been dropped. A SO or SI could be dropped by substringing, assignment to a shorter field, or truncation due to CONVERSE overlay or windowing.

When the field is partially displayed, SO and/or SI are added by MANTIS for display purposes. After a CONVERSE, MANTIS automatically merges the input with the undisplayed parts of the string. The examples below illustrate this process.

**Examples.** In the following examples, the strings are partially displayed at the right edge of the terminal window, so that the left side of the string can be truncated.

1. AB< 1 2 >C <=== text variable input  
     AB< 1 > <=== input partially displayed at CONVERSE  
     AB< 4 > <=== user changes 1 to 4  
     AB< 4 2 3 >C <=== text variable after CONVERSE
2. AB< 1 2 3 >C <=== text variable input  
     AB< 1 > <=== input partially displayed at CONVERSE  
     A< 1 >B <=== user added B after 1  
     A< 1 >B< 3 >C <=== text variable after CONVERSE

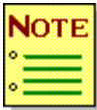
In the following examples, the strings are partially displayed at the left edge of the terminal window, so that the right side of the string can be truncated.

3. AB< 1 2 3 >C <=== text variable input

< 2 3 >C <=== input partially displayed at CONVERSE

C< 2 3 > <=== user changed

A<>C< 2 3 > <=== text variable after CONVERSE



---

If your terminal is a non-DBCS terminal, MANTIS displays DBCS screen fields as “%%%%%%%% ...” with the protected attribute.

For example: A= G"a b c < 1 2 > d e < 3 < f g "

Displays as: A= G"a b c % % % % % d e % % % % f g " on a non-DBCS terminal.

---



## Mixed-data expressions

In using mixed-data expressions, note the following rules as described below:

- ◆ Expressions are rounded down to the even number of bytes in the DBCS substring.
- ◆ MANTIS maintains balanced SO/SI throughout truncation.
- ◆ Leading and trailing null DBCS(SO/SI) are removed in assignment and concatenation.
- ◆ POINT function and substring refer to character positions not byte positions.

### MIXMODE statement

This statement can be set to OFF or ON and controls the handling of mixed-data. MIXMODE ON sets the program in mixed-data mode. MIXMODE OFF sets the program in non-mixed-data mode. Optimum efficiency is obtained with MIXMODE OFF when the program does not handle mixed-data. Do not specify MIXMODE ON unless you are using text expressions containing shift codes. For more information, see "MIXMODE" on page 329.

### Literals and variables

MANTIS stores mixed-data as either literals or TEXT variables.

```
00020 TEXT A(80),B(80)
00030 A=G"abc< 1 2 >de< 3 >fg"
```

The SO/SI bytes occupy one character position (byte) each. Therefore, you must consider the defined length as specified in the TEXT statement. TEXT variables need one byte for each SO and SI, one byte for each SBCS character, and two bytes for each DBCS character. For example:

```
00010 MIXMODE ON
00020 TEXT ALPHA(3)
00030 ALPHA=G"< 1 >" <==== The variable ALPHA will contain the
                                null value with current length of zero.
```

SO/SI take one position each leaving 1-byte (an odd number of bytes). MANTIS truncates the odd byte and leaves 0 bytes for DBCS data. In this example, ALPHA must be at least four characters to hold the data in statement 30. Any TEXT field must be at least four characters long to hold any non-null DBCS characters.

MANTIS keeps SO/SI pairs balanced in truncation. For example:

```
00010 MIXMODE ON
00020 TEXT BETA(7)
00030 BETA=G"< 1 2 3 4 >" <=== The variable BETA will
                                     contain < 1 2> with
                                     current length of six bytes (two
                                     characters).
```

MANTIS removes the empty SO/SI pairs on both sides. For example:

```
00010 MIXMODE ON
00020 TEXT GAMMA(20)
00030 GAMMA=G"<><>< 1 ><>< 2 ><>" <=== The variable GAMMA
                                             will contain 1><>< 2>
                                             and current length of ten
                                             bytes (two characters).
```

## Concatenation

You can concatenate two text or mixed-data variables or literals by using the plus (+) operator. For example:

```
00010 MIXMODE ON
00020 TEXT ONE(12),TWO(12),THREE(24)
00030 ONE=G"< 1 2 >"
00040 TWO=G"< 3 >AB"
00050 THREE=ONE+TWO
```

THREE will contain < **1 2 3** >AB.



---

The unnecessary null text (><) is removed.

---

## Deconcatenation

You can deconcatenate or remove a portion of a mixed-data variable using the minus (-) operator. For example:

```
00010 MIXMODE ON
00020 TEXT ONE(24), TWO(24), THREE(24), FOUR(24)
00030 ONE=G" < 1 2 3 >AB< 4>"
00040 TWO=ONE-G" < 2 >"
00050 THREE=ONE-G"AB"
00060 FOUR=ONE-G< 3 >A"
00070 X=POINT(ONE-G" < 2 >" )
00080 Y=POINT(ONE-G"AB" )
00090 Z=POINT(ONE-G" < 3 >A" )
```

TWO will contain < 1 3 >AB< 4 >

X will contain 2

THREE will contain < 1 2 3 4 >

Y will contain 4

FOUR will contain < 1 2 >B< 4 >

Z will contain 3

## Subscripts

You can use subscripts to reference substrings of a mixed-data variable. The following example illustrates how this process works:

```
00010 MIXMODE ON
00020 TEXT MSG(50)
00030 MSG=G"ABC< 1 2 3 >DE< 4 5 >"
```

The subscripts below show how you can reference substrings of the variable listed above. When specifying character positions for mixed-data variables, MANTIS does not count shift codes. The subscripts refer to CHARACTER positions, not BYTE positions.

Example	Results	Comments
MSG	"ABC< 1 2 3 >DE< 4 5 >"	SIZE(MSG) = 10 (character count).
MSG(9)	"< 4 5 >"	
MSG(5,5)	"< 2 >"	
MSG(6,8)	"< 3 >DE"	
MSG(11)	" "	Beyond end of current length.
MSG(-1)	"< 5 >"	Last character.
MSG(-5,-4)	"< 3 >D"	

## Literals and mixed-data expressions

MANTIS provides a Mixed Text literal, G", to support Shift-in and Shift-out strings. The following example shows how to use this literal:

```
00010 KANJI ALPHA  
00020 ALPHA=G"< 1 2 3 >"
```

When you assign the Mixed Text literal to a KANJI or DBCS variable, MANTIS removes the SO/SI on both sides. You can specify a mixed literal more quickly using the Mixed Text literal than by using a Kanji literal (K" .... "). To illustrate this, instructions are provided below for using a Kanji literal and a Mixed Text literal:

- ◆ **Kanji literal.** You must enter K" ..... " and then press ENTER.

The terminal will make the quoted string into a DBCS field, and you can then enter the DBCS characters.

- ◆ **Mixed Text literal.** You can use the keyboard to shift modes without an additional I/O. Use G" to assign values to either TEXT (mixed) or DBCS fields.

## Built-in functions

Several of the MANTIS functions are affected by mixed-data support, and there are some new functions that help provide this support. The syntax and function of these built-ins are described throughout this section. For more information, see the individual functions in “[MANTIS programming language](#)” on page 75.

### SIZE

```
SIZE(t, "BYTlength")
```

“BYT” returns the current bytelength of the specified string variable (including shift codes). If a second parameter on SIZE is not specified, the current number of characters is returned. If *name* is not a TEXT variable, then a program fault occurs.

```
00010 MIXMODE ON
00020 TEXT ALPHA(20),BETA(20)
00030 KANJI GAMMA(20)
00040 ALPHA=G"< 1 2 >A"
00050 BETA=G"A<>B"
00060 GAMMA=G"< 1 2 >"
00070 A=SIZE(ALPHA,"BYT")
00080 B=SIZE(BETA,"BYT")
00090 C=SIZE(GAMMA,"BYT")
```

A, B, and C have values 7, 4, and 6.

If a second parameter is not specified, the current number of characters is returned. In the previous example, if you change the values to:

```
00070 A=SIZE(ALPHA)
00080 B=SIZE(BETA)
00090 C=SIZE(GAMMA)
```

A, B, and C have values of 3, 2, 2.

You cannot specify arrays as *name*.

```
00070 A=SIZE(ALPHA(5), "BYT")
```

The result (A) is unpredictable.

SIZE(name, "BYT") and TEXT name(*n*) are the two instances where the number of bytes is considered, not the number of characters.

## POINT

```
POINT( t1 +- t2 )
```

When calculating the positions for a mixed-data variable, MANTIS uses character positions and does not count shift codes.

```
00010 MIXMODE ON
00020 TEXT ALPHA(20), BETA(20)
00030 ALPHA=G"< 1 2 3 >"
00040 BETA=G"AB< 1 2 >C"
00050 A=POINT(ALPHA-G"< 2 >")
00060 B=POINT(BETA-"C")
```

A and B will have the values 2 and 5.

## MIXM

```
MIXM(name)
```

MIXM converts a DBCS field to a mixed text field as shown below:

```
00010 MIXMODE ON
00020 TEXT ALPHA(20)
00030 DBCS GAMMA(20)
00040 GAMMA="K" 1 2 "
00050 ALPHA=MIXM(GAMMA)
```

ALPHA contains G"< 1 2 >".

## MIXD

MIXD(t)

MIXD extracts the DBCS data from mixed-data as shown below:

```
00010 MIXMODE ON
00020 TEXT ALPHA(20)
00030 KANJI GAMMA(20)
00040 ALPHA="A< 1 >BC< 2 >"
00050 GAMMA=MIXD(ALPHA)
```

GAMMA contains K“1 2”.

## MIXT

MIXT(t)

MIXT extracts the SBCS text data from mixed-data as shown below:

```
00010 MIXMODE ON
00020 TEXT ALPHA(20),BETA(20)
00030 ALPHA="A< 1>BC< 2 >"
00040 BETA=MIXT(ALPHA)
```

BETA contains “ABC”.



## Statements and commands

Several statements and commands are affected by mixed-data support. This section explains the modified functionality. For more information about individual statements or commands, see “[MANTIS programming language](#)” on page 75.

### LET

If truncation occurs, SO/SI pairs are automatically balanced in mixed-data variables as shown below:

```
00010 MIXMODE ON
00020 TEXT ALPHA(6), BETA(8)
00030 ALPHA=G"ABC< 1 >"
00040 BETA=G"ABC< 1 2 >"
```

The variable ALPHA will contain ABC because there is not room for DBCS data and shift codes. The variable BETA will contain “ABC< 1 >” because the DBCS is truncated to an even number of bytes and a trailing SI is added.

MANTIS only checks for SO/SI when MIXMODE is ON. Otherwise truncation occurs and substringing will NOT maintain SO/SI balancing. However, if MIXMODE OFF is specified as shown below, the variable ALPHA now contains “ABC< 1” and BETA contains “ABC< 1 ”. These variables may result in an abend when they are displayed.

```
00010 MIXMODE OFF
00020 TEXT ALPHA(6), BETA(7)
00030 ALPHA=G"ABC< 1 >"
00040 BETA=G"ABC< 1 2 >"
```

When subscripts are specified, MANTIS places the SO/SI pairs in the proper location as shown below:

```
00010 MIXMODE ON
00020 TEXT ALPHA(15), BETA(15), GAMMA(14)
00030 ALPHA=G"A< 1 2 >BCDE< 3 >"
00040 BETA=ALPHA
00050 GAMMA=ALPHA
00060 BETA(6,6)=G"< 4 >"
00070 GAMMA(4,6)=G"< 4 >"
```

BETA will contain A< 1 2 >BC< 4 >E and GAMMA will contain A< 1 2 4-  
--->E,

where - represents a 1-byte blank.

In the previous example, if you changed the following values:

```
00060 BETA( 3 , 4 ) = " "
00070 GAMMA( 6 , 8 ) = " "
```

the resulting variable BETA would contain A< 1---->CDE, and GAMMA would contain

A< 1 2 >BC---, where - represents a 1-byte blank.

## SHOW

If the mixed-data is extended to the next display line, SO/SI must be paired for each line of the display as shown below:

```
00010 MIXMODE ON
00020 TEXT ALPHA(75)
00030 PAD ALPHA"A"
00040 SHOW A+G"< 1 2 >"
```

The resulting lines would occur:

AAAAAAAAAAAAAAAAAAAAAA . . . . .AA12

## ATTRIBUTE

```
ATTRIBUTE(map,field)={ "MIX" | "NOMIX" }
```

You can modify the SO/SL screen design field attribute.

Field attribute	Description
"map"	Screen name.
"name"	Screen field name.
"MIX"	Enables you to enter DBCS in affected fields.
"NOMIX"	Disables DBCS entry.

If the field name is omitted, all text fields are affected.

The changed attributes will remain in effect until you change them again. The attribute(map,field)=RESET will not reset a "MIX" or "NOMIX" specification. The last set value remains when a RESET is issued. For example:

```
00070 ATTRIBUTE(MAP,F1)="MIX"
00080 ATTRIBUTE(MAP)="RESET"
00090 CONVERSE MAP          <===  Field F1 has MIX attribute
00100 ATTRIBUTE(MAP,F1)="NOMIX"
00110 ATTRIBUTE(MAP)="RESET"
00120 CONVERSE MAP          <===  Field F1 has NOMIX attribute
```

## PAD

You can use mixed-data which represents the pad character (SBCS or DBCS) as shown below:

```
00010 MIXMODE ON
00020 TEXT ALPHA(10)
00030 PAD ALPHA G"< 1 >"
```

ALPHA will contain < 1 1 1 1 >.

```
00010 MIXMODE ON
00020 TEXT BETA(10)
00030 BETA="ABCDEF"
00040 PAD BETA(3,5) G"< 1 >"
```

BETA will contain AB< 1 1 1 >.

## UNPAD

You can use mixed-data which represents the unpad character (SBCS or DBCS) as shown below:

```
00010 MIXMODE ON
00020 TEXT ALPHA (10)
00030 ALPHA=G"ABC< 1 1 >"
00040 UNPAD ALPHA G"< 1 >"
```

ALPHA will contain ABC.

```
00010 MIXMODE ON
00020 TEXT BETA(10)
00030 BETA=G"AB< 1 1 >C"
00040 UNPAD BETA(3,4) G"< 1 >"
```

BETA will contain ABC.

# Glossary of terms

## **| (vertical bar)**

Default blank fill character used in Screen Design. Because MANTIS interprets a blank space as a new field, the blank fill character is used to connect words or letters in heading fields. Using the blank fill character optimizes transmission and screen storage.

The vertical bar character is also used as the first character of a comment.

## **\* (asterisk)**

Entered on parameter entry panels and on the command line as a wildcard character to represent an indefinite number of characters in a generic pattern of program names (e.g., CUST\*). In addition, the asterisk is also supplied by the Compose action in the COMPONENT statements and CEND statements for composed programs, for example, |\*COMPONENT and |\*CEND.

## **@ (at sign)**

The at sign is the default character that you append to a source program name to differentiate it from a composed program name or a component name. When the at sign is appended to a source program name, the Compose action assembles and replaces a composed program with the same name as the source name without the at sign. For example, if your source program is CUST\_BROWSE@, and you issue the Compose action on the source program, the resulting composed program name is CUST\_BROWSE. Note that your Master User may change the differentiator character to a different character for your environment. In addition, the at sign is also coded in the SOURCE statement of an executable program (|@SOURCE) to nominate MANTIS source code changes. Coding a COMPONENT statement with the at sign (|@COMPONENT) nominates that component for the Decompose action.

## **attribute**

Specific characteristic(s) assigned to the fields in a screen during a screen design session. For example; field name, field length, vertical and horizontal repeats, color, highlight, protected, unprotected. Most field attributes can also be set in programming mode using the ATTRIBUTE statement.

## **background task**

A CICS MANTIS task running which is not attached to a terminal device.

## **BIG**

A data type occupying a double-precision floating point variable.

## **bind**

An action that creates an HPO-bound version of a MANTIS program.

## **CEF**

See Component Engineering Facility.

## **CEND statement**

The statement that CEF generates to mark the end of individual component code in a composed program. CEND (component end) statements are generated only if the COMPONENT statements in the composed program are commented by specifying COMMENTS=YES (parameter in the CSIOPTNS statement) or by setting the Function Option "Component stmt?" to Y (yes) on the COMPOSE Program Entry panel.

## **chain**

A generic data structure which has a beginning and an end, is normally searched sequentially, and elements can be inserted or deleted at any point. For example, a program chain.

CHAIN is also a program statement that transfers control to another program.

**comments**

Mantis program lines or ends of lines containing comment information that is not executed as part of the program.

A comment is also a keyword parameter in the CSIOPTNS statement for a source program. The format is COMMENTS=YES or COMMENTS=NO. If YES, the COMMENTS parameter comments the COMPONENT statements in a composed program and generates a |\*CEND statement to mark the end of individual component code. If NO, components in the composed program are framed by their first and last statements only.

**COMMIT points**

Point at the end of a Logical Unit of Work (LUW) (sometimes called a synchronization point) where MANTIS automatically generates a COMMIT if it encounters any uncommitted updates. COMMIT points can also be specified by the user.

**complex variable**

A variable that contains and defines other variables, for example, FILE, ACCESS, TOTAL, VIEW, INTERFACE, or SCREEN variables.

**component**

A MANTIS subroutine that is common to more than one program. Components can be used and reused as necessary as building blocks of code throughout an application. Components can be framed by ENTRY and EXIT statements (although this is not required), and they are stored in a library like other MANTIS programs. Components are identified in source programs by the COMPONENT statement. When the Compose action is issued on the source program, the COMPONENT statement is expanded into component code in the resulting composed program.

**Component Engineering Facility (CEF)**

The MANTIS facility that allows you to include reusable components as the building blocks in a structured and modular design. In CEF, source programs and their components are assembled into composed programs that can be edited and executed. In addition, executable programs can be decomposed into source code and component code. CEF uses the actions of CEF Check, Compose, Decompose, CREF (Cross Reference), and the Bill of Materials List.

## **compose**

An action that assembles a MANTIS source program and its COMPONENT statement(s) into a composed program with expanded component code.

## **composed program**

A MANTIS program containing source code and component code that is the result of issuing the Compose action on a source program.

Composed programs are executable programs that can be edited and executed. Composed programs also have a source program version of MANTIS source code and COMPONENT statements on which the Compose action was issued. See executable program and source program.

## **data area**

Part of a MANTIS program where definitions and values of symbolic names are kept. Each variable has its own associated data area within the larger memory allocation. Also known as the Data Work Area (DWA). See “[DWA](#)” on page 601.

## **data block**

An entry in the DWA that completely defines a variable or complex variable. It consists of a DATA HEADER followed by 0-*n* DATA ELEMENTS.

## **data element**

A single variable within a DATA BLOCK, for example, a single SMALL/BIG/TEXT/KANJI entry.

## **data position characters**

Type of edit characters used in an edit mask to specify that the mask should be zero-filled. For example, to designate a field to enter social security numbers, you would specify `Z## ###-####`. When the data 012345678 is entered the field will display as 012-34-5678.



**decompose**

An action that disassembles an executable MANTIS program into individual components and then updates program libraries with source changes and component changes.

**DOLEVEL**

An indication of how many levels of external DOs a program is running under. The first program executed by a CHAIN has a DOLEVEL of zero. Also a built-in function indicating the same.

**domain**

The space in, or the invisible boundary around, a defined screen or field. Domains ensure that the screen and field definitions (including attributes and repeat specifications) made during screen design are retained until they are modified.

**DOSTACK**

A context block used when external DOs are activated. Information relative to the executing program (from TWA) is pushed and popped from this stack as programs are externally executed and exited.

**double byte character set (DBCS)**

Two-byte characters used on Asian language support terminals.

**DWA**

“Data Work Area. A context block containing the data values and definition of MANTIS variables for a program.

**edit characters**

Special characters used to allow flexible formatting of numeric data fields. Edit characters display only when data is entered in a numeric field. For example, to display a field with a dollar amount, use the dollar sign as an edit character. When numbers are entered in the field, the edit character will display; otherwise it remains hidden.

## **edit masks**

Special characters that allow you to format numeric fields to display data in a certain way. Edit masks are mainly used for formatting output fields. For example, to display a list of check amounts in a column, use an edit mask that displays the amounts in the correct, right-justified format.

## **entity (or MANTIS entity)**

Generic name for complex variable descriptions (SCREEN, INTERFACE, FILE), MANTIS programs, and MANTIS internal file data.

## **executable program**

Any program that can be executed in MANTIS. An executable program that is the result of the Compose action, is called a composed program. See composed program and source program.

## **extended attribute support**

Support in Screen Design for advanced attributes available on the 3270 style terminal. This includes color, blinking, underlining, reverse video, and so on.

## **external file view**

Detailed information about the contents and format stored in an external file, such as a VSAM file. A file view allows you to control access to the information by password protecting certain portions of the file data. Defined by the ACCESS statement.

## **field attribute**

Attribute such as color, highlighting, blinking, underlining, and so on defined to a field on a screen.

## **file view**

See “[internal file view](#)” on page 605, “[external file view](#)” on page 602, or “[TOTAL file view](#)” on page 613.

**fill characters**

Any edit mask character (other than a blank and #) that occurs in consecutive positions in a mask, but does not begin in the first position. Fill characters can be used to display columns of numeric information in a particular way, such as a list of check amounts that needs to be right justified.

**fixed position characters**

Any edit mask character (other than # and the sign characters when used as fill or floating characters) that does not occur in consecutive positions in a mask. With a few exceptions, fixed position characters are always displayed in a mask.

**float characters**

Any character (other than blank, #, +, and -) that occupies the first and at least one consecutive position of the edit mask.

**Full Screen Editor (FSE)**

Provides facilities for creating and modifying MANTIS programs using the logical screen support of the Logical Terminal Interface (LTI). FSE is accessed through the Edit Option on the Program Design Facility menu.

**function key**

(1) The program function (PF) keys that issue a specific action. PF keys are displayed at the bottom of panels and can be changed for the duration of the current action. Your Master User can permanently customize PF key settings for each user. Examples of PF keys are F1=HELP, F2=EXHELP, F3=EXIT, and F4=PROMPT. (2)The program function (PF) keys that issue a specific action. PF keys are available for each facility and differ from screen to screen within the facility. Your Master User can permanently customize PF key settings for each user.

## **function key area**

The area at the bottom of your panel where function key numbers and their settings are displayed.

## **header**

The initial part of a simple or complex variable. For a simple variable, it consists of the variable's typing definition. For a complex variable, it consists of typing definition, status variable value, and current context (keys, marks, flags, etc.).

## **heading fields**

Fields defined in Screen Design to specify screen and field names. Heading fields always appear on the completed design exactly as they were entered.

## **help**

The common dialog action for field-specific help. When issued, HELP displays a help panel that explains a specific field (based on cursor position). HELP can also display a help panel for a command or message, or HELP can display the KEYSTEMP panel where you may alter PF keys.

## **horizontal repeats**

Attribute in Screen Design to indicate the number of times a field on a screen is to be repeated horizontally.

## **HPO bind**

The function that creates a new bound version of a MANTIS program.

## **HPO unbind**

The function that replaces the bound version of a MANTIS program with the unbound version.

**immediate mode statement**

A statement entered without a line number in the Line Editor, or on the command line of the Full Screen Editor, indicating it should be interpreted and executed immediately, and not become a part of the program.

**index**

An area of the Variable Work Area (VWA) that maps user word numbers to DWA offsets for the corresponding variables.

**INTERFACE**

A complex data type that defines an area used by a non-MANTIS program when CALLED. Also used to refer to the program as the object of a CALL statement.

**interface area**

A context block used to pass data between MANTIS and an interface program.

**internal file view**

Same as MANTIS “[file view](#)” on page 602; that is, a file defined by a MANTIS user and residing on the MANTIS cluster.

**Kanji**

A generic term indicating a 16-bit text data type, and specifically a Japanese language 16-bit character set.

**leveling**

Use of the LEVEL option on definition or action statements for complex variables. This indicates the simple variables are subscripted by an order of the level specification on the definition statement. For example, FILE REC(“MISC”, “PASSWORD”, “10”) indicates a dimension of 10 for all simple variables defined in “MISC” file. GET REC LEVEL=8 indicates that a successful GET fills the eighth element of each of these arrays.

## **line commands**

FSE commands that affect the line(s) on which they are entered. Line commands include editing commands (move, copy, etc.) as well as destination commands (after, before, etc.).

## **line editor**

The single-line program editing facility used in prior releases of MANTIS. It is still available by running "CONTROL:LINE\_EDIT". Since the Line Editor does not update the Entity Profile Records, programs should be written using the Full Screen Editor.

## **logical terminal**

An abstract device with 255 rows and 255 columns and a superset of all terminal features. Screen I/O is directed to the logical terminal, and subsets of dimensions and features are directed to the physical terminal device, depending upon its actual capabilities.

## **logoff**

A common dialog action that lets you exit from MANTIS. If you are working in the Full Screen Editor when you issue LOGOFF, your changes are saved.

## **LTI**

Logical Terminal Interface. The facility which directs and controls input and output to physical and logical terminal and printer devices.

## **map**

Commonly used as a synonym for screen or panel.

## **map set**

The collection of all screens displayed to the logical terminal at any one time. A screen is added to the map set by a CONVERSE mapname SET or CONVERSE mapname WAIT. A map set is cleared by CLEAR or CONVERSE without SET or WAIT.

## **Master User**

Person or persons designated to perform administrative functions for an installation site. The Master User can set up user profiles, specify which users can use which facility programs, alter sign-on and termination, set printing specifications, establish system security, edit text of MANTIS and Kanji messages, maintain files and codes, display program statistics, check bound programs, transfer entities, capture data from a background task, share frequently used programs among users, and run reports.

## **menu**

A common dialog action that lets you return to the MANTIS Facility Selection menu. If you are working in the Full Screen Editor when you issue MENU, your changes are saved.

## **mixed-data type support**

Support allowing both EBCDIC (for example, English) and DBCS (Asian language) to reside concurrently in a variable to indicate whether EBCDIC or DBCS characters are present. Set by the SO/SI attribute in Screen Design and by the MIXMODE ON statement in programming mode.

## **multiple buffering**

See “[leveling](#)” on page 605.

## **nominate**

Marking COMPONENT statements in an executable or composed program by replacing the asterisk (\*) with the at sign (@), for example, |@COMPONENT. Nominating a component indicates to the Decompose action that the component code was modified, and it is decomposed and updated on your user library. If MANTIS source code changes, the SOURCE statement in the executable program must be nominated for the Decompose action to recognize source code changes, for example, |@SOURCE. You may make changes to the source code without changing any components.

## **nucleus**

Procedures and modules of MANTIS that direct the execution of MANTIS programs and are environment-independent.

## **numeric data**

BIG and SMALL variables or expressions or function outputs.

## **numeric fields**

Fields defined for the input and output of numeric data only.

## **opaque map**

Attribute in the Screen Design Library Facility that allows a screen (map) to be opaque (rather than transparent) when it is conversed. This attribute can only be set from the Screen Design Library Facility menu.

## **panel**

A grouping of information arranged in a particular design on a screen. For example, menu panels, list panels, parameter entry panels, and information panels.

## **p-code**

An intermediate code version of the MANTIS program statements that is used to direct program interpretation (execution) and is translated to and from MANTIS statements.

## **PREFIX**

Specification on a complex variable definition where each component simple variable has the complex variable name appended to the beginning of the generated variable name. For example, SCREEN MAP("SCR1",PREFIX) generates a variable definition for MAP\_OPTION when OPTION is a field defined in the screen, "SCR1".



**procedure**

A contiguous set of source code comprising a logical unit. A procedure is delineated by an ENTRY-EXIT statement pair.

**programming mode**

A state of a MANTIS task characterized by executing under the Full Screen Editor or the Line Editor.

**prompt**

A common dialog action that displays the current list of all valid common dialog actions and commands for a panel. The Prompt list includes common dialog actions (common to panels) and function commands (specific to the current function only). You may select an action or command with the selection character (/). When you exit from the Prompt list, your selection is executed.

**prompter**

MANTIS entity for creating online help screens and documentation.

**prototyping**

Creating a working model of a data processing system that reflects system requirements and that can be demonstrated and refined as necessary.

**PWA**

Program Work Area. A context block used to contain the contents of an executing program (the p-code).

**record layout**

Provides the format in which the data in a file is stored and transmitted. You can associate an existing record layout to other file designs.

## **REPLACE statement**

A statement coded in the source program for the Component Engineering Facility (CEF) that names the library, program, password, and description to be created or replaced as the executable program by the Compose action.

## **reserved word**

A word that cannot be used for user variable names, such as PAD, FILE, SCREEN, and so on.

## **running (mode)**

A state of a MANTIS task characterized by executing without an editor involved. For example, a program invoked by a CHAIN from a facility menu.

## **screen**

A grouping of information arranged in a particular design. The MANTIS Facilities are made up of a series of panels. You can create your own panels for application programs using the Screen Design Facility.

## **semi-reserved word**

A user word which has specific meaning in some way, for example, in the MANTIS DL/I interface. These user variables are used in a prescribed manner to communicate with MANTIS components.

**sequence**

A keyword parameter in the CSIOPTNS statement for a source program. The SEQUENCE parameter lets you specify how the line numbers in a composed program are sequenced before the program is replaced. The system default value is SEQUENCE 10,10.

**SETPRAY**

Another name for the MANTIS Cluster, and its DDNAME in CICS.

**sign character**

Includes the plus (+) and minus (-) signs and the Credit (CR) or Debit (DR or DB) signs. The sign characters can only be used for displaying the output of numeric fields.

**simple variable**

A scalar or array defined by the TEXT, BIG, SMALL, or KANJI statements, or as a result of executing a complex variable statement (which implicitly defines it). See “[complex variable](#)” on page 599.

**skeleton programs**

Generic programs supplied with MANTIS for modification by users, including sample browse programs, entry programs, menu programs, and so on.

**SMALL**

A data type occupying floating point singleword precision variable. Also a reserved word and statement.

**source program**

A MANTIS program of source code and at least one COMPONENT statement. Source programs are not executable. The Compose action is issued on a source program to assemble (compose) it into a composed program of source code and expanded component code that you can edit and run. See composed program and executable program.

## **SOURCE statement**

A Component Engineering Facility (CEF) statement that is coded in an executable program to name the library, program, password, and description of the source program to be created or replaced by the Decompose action.

## **SQL bind**

For DB2 environments only: Static: Places information about a program's SQL statements and their host variables into an internal file to create an SQL support module for static execution of the program. Extended Dynamic: Dynamically creates a DB2 for VSE and VM (formerly SQL/DS) Access Module for the program, saves information about SQL statements and host variables, and makes the program immediately executable at the end of the bind.

## **statement**

The smallest executable unit of a MANTIS program.

## **STATUS**

(1) An attribute of some MANTIS entities (USER, FILE), that indicates whether or not the entity can be used. "ACTIVE" indicates it is available, anything else, not. (2) A value associated with complex data entities to indicate for files, PF12 for screens. (3) A feedback mechanism from certain files or database systems (VSAM, TOTAL, DL/I) in which MANTIS receives the result of the last operation against it.

## **terminate (exit)**

A common dialog action that terminates the current function and returns a higher level function. For example, exiting from the Update Field Specifications function takes you back to the Screen Design Facility menu.

## **TEXT**

A character data type containing an EBCDIC (English) text string.

**TOTAL file view**

Detailed information about the contents and format stored in a TOTAL file. A file view allows you to control access to the information by password protecting certain portions of the file data.

**TPI**

Teleprocessing Interface. A module which contains code specific to the host environment, such as CICS, TIS/CM, IMS/DC, OS-BATCH.

**Transfer file**

A VSAM file used by the Transfer Facility to hold MANTIS entities or data for sharing between users or systems.

**TWA**

Transaction Work Area (or Task Work Area) from CICS. Main context block used throughout MANTIS and TPI to hold information about the user, the task, the executing program, work areas, and other global variables. Also contains addresses to all major context blocks. Environment independent.

**unbind**

An action that replaces the HPO-bound version of a MANTIS program with the unbound version.

**uppercase**

Attribute in Screen Design to indicate whether data entry text fields are translated to uppercase characters or remain as the user entered it. This setting is overridden by the ATTRIBUTE statement and disregarded if the terminal does not support uppercase. Uppercase is also available in Prompter Design if terminal support is available.

### **user code**

A code that is a part of the MANTIS cluster VSAM key. This is used to identify records which belong to a given user ID. User codes less than 16 are reserved for Cincom use only.

### **user word**

The name of a MANTIS variable in a MANTIS program. Also known as a USER VARIABLE or VARIABLE. This includes all simple and complex data types.

### **vertical repeats**

Attribute set in Screen Design to indicate the number of times a field is to repeat vertically on a screen.

### **vocabulary**

Set of all user words active for a program. Also used to refer to the area where the text representation of these words are kept.

### **VWA**

Vocabulary Work Area. A context block which contains the INDEX of the data areas (DWA) and the vocabulary area.

### **window mode**

Mode in MANTIS that allows you to view a screen design that is larger than the physical terminal.

# Index

)

semicolon ( 22

## A

ABS

summary 76

ABS 86

absolute value See ABS

ACCESS 87–92

binding consideration 88

code example 92

PREFIX 89

summary 76

accessing

facilities 30

accessing files See FILE

accessing interfaces See  
INTERFACE

accessing prompters See  
PROMPT

active map 158

definition 561

adding new records See INSERT

addition See POINT

angle See ATN

apostrophe See single quote

arithmetic expression 45–49, 94

operands 45

operators 46–49

array

allocating one dimensional 416

arithmetic 45

assigning a value 308

definition 27

naming 415

storage 28

array variables See SMALL

ASI See Attribute Status Indicator

ASI (Function) 93

summary 76

Asian language support See  
DBCS

asterisk

double ( \*\* ) 23

single ( \* ) 23

at sign (@)

composed program 385

nominating components 23

ATN 94

ATTRIBUTE (Function) 95

code examples 102

general considerations 100

restriction 97, 99

summary 76

syntax 95

ATTRIBUTE (Statement) 102–33

CONVERSE 104

cursor positioning 120

mixed data 595

PRINTER 106

RESET 102, 120

restriction 104, 107

summary 76

syntax 102

TERMINAL 108

Attribute Status Indicator 93

attributes

abbreviations 110–12

affecting map sets 566

color 126

field

considerations 560

listed by type 110–12

multiple 113

restrictions for numeric 113

automatic mapping 214

advantages of 33–35

clearing variables 147

code examples 542

compared to COBOL moves 35

definition 33

leveling 541

standard names 35

AUTOSKIP 118

**B**

- background task
  - CSOL (log) file
    - abend codes 364
    - errors 363
  - Non-MANTIS 366
  - termination 363
- Batch
  - COMMIT 150
  - PRINTER restrictions 371
  - status messages 525
- BIG 134–35
  - summary 76
- blank-fill character 560
  - AUTOSKIP 118
- BLINK 118
- BOTTOM LINE ENTERABLE 126
- BOXED 119
- BREAK
  - summary 76
- BREAK 136
- BRIGHT 119
- buffers
  - flushing 149
- built-in function
  - alphabetical list 63–68
  - definition 63
  - listed by type 72

**C**

- CALL 137–38
  - summary 76
- case insensitive compare 60
- centering headings See HEAD
- CHAIN 139–43
  - best conditions for use 546
  - External DO 536
  - I/O accesses 537
  - summary 76
  - uppercase translation 142
  - work area example 536
- CHAIN
  - without LEVEL 141
- character set 22
- CHR 144
- CICS MANTIS
  - function status messages 524
- CLASS
  - selecting print mode 108

- CLEAR
  - array 146
  - HEAD 270
  - summary 77
- CLEAR 145–48
- CMS
  - status messages 525
- colon (
  - ) 22
- color attributes 126
- comma ( , ) 22
- command line 556
- commands
  - listed with mode 76–85
- commands
  - affected by mixed data 593
- comment
  - characters 23
  - indicating in programs 39
- COMMENTS= 166
- COMMIT
  - Batch mode 150
  - summary 77
- COMMIT 149–52
- complete unit of work See COMMIT
- complex statements
  - automatic mapping 214
  - defining 541
- COMPONENT 153–56
  - best conditions for use 546
  - notation requirements 155
  - summary 77
- components for decompose See SOURCE
- Compose action 153
  - forcing 167
- compose options See CSIOPTNS
- composed executable program 153
- conditional execution See WHILE-END. See WHEN-END
- conditional repeat See NEXT
- continuation
  - restrictions 37
- control
  - return from subroutine 388
- control intervals 549–53



CONVERSE 157–63  
 active map 158  
 automatic DEQUEUE 161  
 changing maps passive to  
   active 162  
 clearing maps 162  
 code example 163  
 FULL DISPLAY 161  
 general considerations 160–62  
 map 563  
 passive map 158  
 resulting actions 160  
 summary 77  
 syntax 157  
 with windowing 569  
 CONVERSE  
   *affect on map sets* 562  
   map  
     examples 564–66  
 COS  
   summary 77  
 COS 164  
 cosine See COS  
 CSIOPTNS 165–70  
   notation requirements 169  
   summary 77  
 current date See DATE  
   (Function)  
 current language See  
   LANGUAGE (Statement)  
 current password See  
   PASSWORD  
 current printer setting See  
   PRINTER (Function)  
 current release See RELEASE  
   (Function)  
 current terminal ID See  
   TERMINAL  
 current time See TIME (Function)  
 CURSOR 120, 171–75  
   CLEAR 173  
   general conderations 173  
   location 171  
   restriction 171  
   summary 77  
   syntax 171  
 cursor positioning  
   automatic reset 120

## D

dash See minus sign  
 data  
   passing between programs 142  
   storing  
     in SQLCA 424  
     SQL in MANTIS 432  
   storing SQLDA in MANTIS 429  
   transfer  
     from SQLDA to MANTIS 426  
     SQLCA to MANTIS 420  
   transferring 527  
 data area available See  
   DATAFREE  
 Data Work Area 176  
   allocating 531, 536  
   size limits 533  
 DATAFREE 176  
   summary 77  
 DATE (Function) 177–78  
   summary 77  
 DATE (Statement) 179–80  
 DBCS  
   considerations 29  
   data 62  
 DBCS (Statement) 181–82  
 DBCS data to mixed data See  
   MIXM  
 DBCS from mixed data See  
   MIXD  
 DBCS. See Double Byte  
   Character Set  
 deadlocks 211, 550–53  
   rules for avoiding 553  
 debugging 452  
 debugging 506  
 defining text variables See TEXT  
 DELETE 181–202  
 DELETE (External File)  
   automatic RESET 188  
   general considerations 185–88  
   key 184  
   partial deletion 188  
   syntax 183  
 DELETE (External File) 183–88  
 DELETE (MANTIS file) 181–83  
   syntax 189

DELETE (MANTIS File)  
  automatic RESET 192  
  key 189  
  partial deletion 192  
  returned statuses 192  
  TRAP 192  
DELETE (Personal computer file)  
  189–96  
  syntax 194  
DELETE (RDM Logical view) 197  
DELETE (TOTAL File view) 200–  
  202  
DEQUEUE  
  summary 78  
DEQUEUE 203–5  
  restrictions 204  
DETECTABLE 121  
dimensions 134  
  arrays 415  
  specifying for text variables 457  
  terminal 121  
  variables 415  
displaying data See SHOW  
displaying intensity 119  
dissimilarity debugging 513  
DO  
  affect on performance 207  
  summary 78  
DO 206–8  
DOLEVEL  
  maximum 532  
DOLEVEL 209  
  summary 78  
double quotes (" ") 22  
  REPLACE 385  
dynamic offset See row,column

## E

E (Function) 210  
  summary 78  
editing checks 578  
editing numeric expressions See  
  FORMAT  
ENQUEUE  
  released by CHAIN 142  
  restrictions 212  
  summary 78  
ENQUEUE 211–12

ENTRY-EXIT 213–16  
  CHAIN 142  
  COMPONENT 154  
  DO 207  
    summary 78  
equal sign (=) 23  
error tracking See TRAP  
exclamation point ( ! ) 23  
EXEC\_SQL-END 217  
  summary 78  
executing a new program See  
  CHAIN  
executing current program See  
  RUN  
executing statement blocks See  
  NEXT  
execution level 209  
EXIT  
  command 219  
  summary 78  
exiting  
  loops 136  
exiting a unit of logic See RESET  
exiting conditional statements  
  See BREAK  
EXP  
  summary 78  
EXP 220  
exponents See EXP  
extended status messages 521  
External DO 527–49  
  best conditions for use 545  
  CALL 528  
  calling subroutines 544  
  CHAIN 533–37, 533–37  
  code example 536  
  code examples 548  
  COMPONENT 528  
  debugging 545  
  definition 528  
  entity definition 543  
  I/O reduction 537  
  Internal DO 533–37  
  modularization 540  
  passing parameters 104, 531  
  PERFORM 528  
  PROGRAM 538  
  programming guidelines 537–  
    49  
  supported by reserved words  
    529  
  terminal I/O 543

external file  
     status messages 521  
 external routine level See  
     DOLEVEL  
 external routines  
     command placement 549  
 external subroutine  
     naming 373

## F

Facility Selection Menu 30  
 FALSE  
     summary 78  
 FALSE 221  
 field  
     cursor specifications 120  
 field characteristics See  
     ATTRIBUTE (Statement)  
 field dimensions See SIZE  
 field modification See MODIFIED  
 field status See ASI  
 FILE  
     binding consideration 222  
     summary 78  
 FILE 222–25  
 file status See FSI (Function)  
 files  
     VSAM 550  
 filling variables See PAD  
 FORCE 167  
 FOR-END 226–29  
     summary 78  
 FORMAT  
     summary 78  
 FORMAT 230–31  
 freeing resources See  
     DEQUEUE  
 freeing storage See RELEASE  
     (Statement)  
 FSE See Full Screen Editor

FSI See Function Status  
     Indicator  
 FSI (Function) 232  
     summary 79  
 FULL DISPLAY 121, 560  
 Full-Screen Editor  
     commands 516  
 Full-Screen Editor  
     definition 32  
 Function Status Indicator 232–  
     33, 232–33  
     function format 521  
 functions  
     built in  
         affected by mixed data 590  
         listed by type 63–68

## G

GET 234  
 GET (External file) 234–39  
 GET (MANTIS file) 249–53  
 GET (Personal computer file)  
     254–55  
 GET (RDM Logical view) 259–64  
 GET (TOTAL File view) 265–69

## H

hash character (#) 22  
 HEAD 270–71  
     summary 79  
 heading fields  
     using mixed data 582  
 HELP 272–73  
     summary 79  
 HIDDEN 119  
 highest field status See VSI  
     (Function)  
 HIGHLIGHT 121  
 holding a resource See  
     ENQUEUE

**I**

- IF-ELSE-END 274–76
  - summary 79
- immediate execution See RUN
- IMS
  - COMMIT 150
- incremental execution See FOR-END
- indentation periods See nesting
  - hierarchy
- input data See OBTAIN
- INSERT 277–93
  - INSERT (External file) 277–79
  - INSERT (MANTIS file) 280–82
  - INSERT (Personal computer file) 283–86
  - INSERT (RDM Logical view) 287–88
  - INSERT (TOTAL File view) 291–93
- INT
  - summary 79
- INT 294
- integer value See INT
- INTERFACE 295–97
  - summary 79
- interface area layout 137
- Internal DO
  - best conditions for use 546
  - code example 533
  - definition 528
  - External DO 533–37
  - I/O accesses 537
- internal storage
  - releasing 380
- invoking background tasks See PERFORM
- invoking interfaces See CALL

**K**

- KANJI 181–82
  - summary 79
- KEEP MAP MODIFIED 122, 566
- KEY 301–2
  - summary 79
- key simulation field 556
- keyboard input See KEY
- KILL 303–4
  - summary 79
- KMM See KEEP MAP MODIFIED

**L**

- LANGUAGE (Function) 305
- LANGUAGE (Statement) 306
- LEFT BAR 122
- LET 308
  - mixed data 593
  - summary 80
- LET (Numeric) 308
  - ROUNDED 309
- LET (Text/DBCS) 312
  - code examples 319
  - general considerations 315
- LEVEL
  - CHAIN 141
- Line Editor
  - definition 32
- line numbers
  - sequencing 168
- locating cursor position See CURSOR
- locating names used See USAGE
- locking a resource See ENQUEUE
- LOG 320
  - summary 80
- logarithm See LOG
- logic verbs
  - list 38
- logical display
  - size 556
- logical interface 555
- Logical Terminal Interface
  - MANTIS 555
- Logical Unit of Work
  - backing out 387
  - completion 149
- logical view
  - current position 324
- logon See sign on
- loop termination See KILL
- loops
  - exiting 136
- LOWERCASE 131, 321–22
  - summary 80
- LTI See Logical Terminal Interface
- LUID 323

**M**

**MANTIS**  
 overview of 21  
**MANTIS** access to SQLDA See  
 SQLDA  
**MANTIS** file  
 status messages 521  
 list 522–23  
 mantissa  
 definition 43  
 map  
 clearing 574, 573–74  
 map set  
 attributes 566  
 building 561–62  
 clearing 146, 574–77  
 examples 575–77  
 CONVERSE 562  
 position of 561  
**MARK** 324  
 summary 80  
 Master User  
 definition 29  
 MDT's See Modified Data Tags  
 memory storage 532  
 message line 556  
 minus sign (-) 23  
**MIX** 123  
 RESET 123  
**MIXD** 327  
 mixed data 592  
 summary 80  
 mixed data  
 in text variable 580  
 literals 585  
 setting mode 329  
 support 123, 579  
 variables 585  
 mixed data  
 expressions 585  
 heading fields 583  
 mixed data mode See MIXMODE

**MIXM** 328  
 mixed data 591  
 summary 80  
**MIXMODE** 329–30, 585  
 summary 80  
**MIXT** 331  
 mixed data 592  
 summary 80  
**MODIFIED** 123, 332–34  
 summary 80  
 modified data tags 122

**N**

naming DBCS variables See  
 DBCS. See KANJI  
 naming designs See SCREEN  
 naming external subroutines See  
 PROGRAM  
 naming interfaces See  
 INTERFACE  
 naming numeric variables See  
 SMALL. See BIG  
**NATIVE LANGUAGE SUPPORT**  
 See NLS  
 natural (e) See E  
 natural logarithm See LOG  
 navigation settings See SCROLL  
 nesting hierarchy 36  
**NEXT** 335  
 summary 80  
**NLS**  
 current language 125  
 language codes 124  
**NO ALARM** 129  
**NO AUTOSKIP** 118  
**NO BLINK** 118  
**NO COLOR** 126  
**NO CURSOR** 120  
**NO FULL DISPLAY** 121  
**NO HIGHLIGHT** 121  
**NO LEFT BAR** 122

NO MIX 123  
NO OVERLINE 126  
NO RIGHT BAR 128  
NO UNDERLINE 130  
nominating entities See  
    REPLACE  
nominating subroutines See  
    COMPONENT  
NON DETECTABLE 121  
NORMAL 119  
NOT 336–37  
    summary 80  
NULL 338  
    summary 80  
number set  
    definition 27  
NUMERIC 339  
numeric attribute  
    restriction 113  
numeric data  
    considerations 27–28, 27–28  
    precision 44  
    storage  
        BIG 42  
        SMALL 42  
    storage 27, 41  
        example 44  
    significant digits 42  
numeric value  
    text string 499  
numeric value of text See VALUE  
numeric variables  
    specifying dimensions 134

## O

OBTAIN 341–42  
    summary 81  
online help See HELP  
opaque map  
    definition 561  
opening external files See  
    ACCESS  
operands 45  
operators 46–49  
OUTPUT 344–45  
    PRINTER (Statement) 371  
        summary 81  
OVERLINE 126

## P

PAD 346–47  
    mixed data 595  
    summary 81  
parameter  
    passing with External DO 104,  
        531  
parentheses ( ) 22  
passive map 158  
    definition 561  
PASSWORD 349  
    summary 81  
PC CONTACT  
    status messages 526  
PERFORM 350–67  
    "BACK" 361–62  
    "EXEC" 358–60  
    "EXTN" 366–67  
    "program" 353–55, 356–57,  
        358–60  
    "XCTL" 356–57  
    summary 81  
PERFORM  
    options 352  
period (.) 22  
physical screen  
    repositioning 571  
PI 367  
    summary 81  
plus sign (+) 23  
POINT 368–70  
    mixed data 591  
    summary 81  
populating number generator See  
    SEED  
populating variables See LET  
pound sign See hash character  
PRINTER (Function) 370  
    summary 81  
PRINTER (Statement) 371  
    Batch restrictions 371  
    summary 81  
    uppercase translation 371  
printing data See OUTPUT  
PROGFREE 372  
    summary 81

- program
  - clearing all data 145
  - invoking 350
  - size limits 410
  - statement count 411
  - suspension 410
- program
  - size considerations 547
- PROGRAM 373–75
  - summary 82
- program architecture 531–32
- program area available See PROGFREE
- program control
  - NON-MANTIS background task 366
  - transfer saving MANTIS context 358
  - transfer without return 356
  - transfer without variables 352
- program control
  - transferring 527
- program execution 391
  - stopping 388
- program loop See SLOT
- program stack
  - releasing 532
- program statement limits See SLICE
- program suspension See WAIT
- program termination See STOP
  - conditions 32
- Program Work Area
  - size limits 533
- programming
  - techniques
    - advanced 527–53
- PROMPT 376–77
  - summary 82
- prompter
  - displaying 376
- PROTECT BOTTOM LINE 126, 560
  - key simulation field 126
- PROTECTED 127

## R

- random number
  - sequencing 398
- random real number See RND
- RDM
  - Status functions 517–20
- RDM logical record field
  - status 93
- RDM logical view 501
- RDM logical view position See MARK
- RDM view retrieval See VIEW
- reading a record See GET
- record
  - replacing 479
- recovering program control See EXIT
- recovering program control See RETURN
- referenced variable 474
- RELEASE (Function) 378–79
- RELEASE (Statement) 380
- releasing TOTAL views See DEQUEUE
- removing data See CLEAR
- removing placeholders See UNPAD
- removing records See DELETE
- repeated execution See UNTIL-END
- REPLACE 383–86
  - summary 82
- replacing records See UPDATE
- reserved words xiii, 515
  - definition 24
  - use in programs 31
- RESET 127
  - MIX 123
  - NO MIX 123
  - summary 82
- RESET (Statement) 387
- RESET MAP MODIFIED 122, 566

resetting keys. *See* CLEAR  
resources  
    controlling 211  
restoring Screen Design values  
    127  
retrieving files *See* ACCESS  
retrieving RDM Views *See* VIEW  
retrieving TOTAL views *See*  
    TOTAL  
RETURN  
    summary 82  
RETURN 388  
REVERSE VIDEO 128  
RIGHT BAR 128  
RMM *See* RESET MAP  
    MODIFIED  
RND 389–90  
    summary 82  
ROUNDED  
    LET (Numeric) 309  
routing data *See* PRINTER  
    (Statement)  
routing output *See* OUTPUT  
row ,column 157  
RUN 391–92  
    partial program 32  
    summary 82

## S

SAF *See* SEND ALL FIELDS  
SBCS from data *See* MIXT  
scalar variables *See* SMALL  
scientific notation 42–44  
    definition 42  
SCREEN 393–95  
    summary 82  
screen design 556–60  
    multiple images 567–68  
    specifying 393  
    using mixed data 581  
screen design  
    work area 556  
screen heading *See* HEAD  
screen overlay 563  
SCROLL 396–97  
    summary 82  
SEED 398  
    summary 83  
SEND ALL FIELDS 129  
SEND MODIFIED FIELDS 129

sequencing program line  
    numbers 168  
setting time format *See* TIME  
    (Statement)  
SGN 399  
    summary 83  
Shift-in 329, 579  
Shift-out 329, 579  
SHOW 400–403  
    Batch restrictions 401  
    mixed data 594  
    summary 83  
SI *See* Shift-in  
sign on  
    MANTIS Sign-On Screen 29  
SIN 403  
    summary 83  
sine *See* SIN  
single quote (') 22  
SIZE 404–9  
    mixed data 590  
    summary 83  
SIZE 514  
size of terminal *See* TERMSIZE  
slash (/) 23  
SLICE 410–12  
    DO 207  
    summary 83  
SLOT 413–14  
    summary 83  
SLOT  
    DO 207  
SMALL 415–16  
    numeric storage 27  
    summary 83  
SMF *See* SEND MODIFIED  
    FIELDS  
SO *See* Shift-out  
SO/SI attribute 581  
SO/SI pair 583  
SOUND ALARM 129  
SOURCE 417–19  
    summary 83  
source program 154  
    returning to library 383  
special characters  
    list 22–23  
SQL statements *See*  
    EXEC\_SQL-END  
SQLCA (Function) 420–23  
    summary 83  
SQLCA (Statement) 424–25



- SQLDA (Function) 426–31
- SQLDA (Function)
  - summary 83
- SQLDA (Statement) 432–50
- SQR 451
  - summary 83
- square root See SQR
- starting user-written programs
  - See PERFORM
- statements
  - executing a block 478
  - listed with mode 76–85
  - repeated execution 226
- statements
  - affected by mixed data 593
- step processing See STOP. See IF-ELSE-END
- STOP 452–53
  - summary 84
- subroutine
  - return control 388
- subroutine boundaries See ENTRY-EXIT
- subscript
  - definition 44
  - examples 53
- subtraction See POINT
- symbolic names 25
  - consistency checking 24
  - definition 24
  - location in a program 495
  - maximum in program 24
  - uppercase translation 24
  - using reserved word 24
- symbolic names in use See USERWORDS

**T**

- TAN 454
  - summary 84
- tangent See TAN
- TERMINAL
  - summary 84
- TERMINAL 455
- terminal output See CONVERSE
- TERMSIZE
  - summary 84
- TERMSIZE 456
- testing cursor location See CURSOR
- testing input See FALSE
- testing negative outcomes See ZERO
- testing values See TRUE
- TEXT
  - summary 84
- TEXT 457–59
- text data
  - storage 50
  - literals 50
- text expression 55–58
  - examples 57
- text string
  - converting to uppercase 493
  - numeric value 499
- text string conversion 230. See UPPERCASE. See LOWERCASE
- text value 473. See TXT
- text variable
  - definition 52
  - mixed data in 580

TIME (Function) 460–61  
    summary 84  
TIME (Statement) 462–63  
    summary 84  
TOTAL  
    avoiding system resource lock  
        up 467  
    loop conditions 470  
    summary 84  
TOTAL 464–68  
TOTAL lock up  
    COMMIT 150  
tracking errors See TRAP  
transferring program execution  
    See DO  
transferring SQL data See  
    SQLCA (Function)  
translation table  
    specifying 124  
translucent map  
    definition 561  
TRAP  
    summary 84  
TRAP 469–71  
TRUE  
    summary 84  
TRUE 472  
TXT 473  
    summary 84

## U

unary operators 48  
unary signs See SGN  
UNBOXED 119  
underline 22  
UNDERLINE 130  
unformatted data  
    assigning values 341  
UNMODIFIED 123  
UNPAD 474–77  
    mixed data 596  
    summary 84  
UNPROTECTED 127  
UNTIL-END 478  
    summary 84

UPDATE 479–92  
UPDATE (External file) 479–85  
UPDATE (MANTIS file) 479  
UPDATE (Personal computer  
    file) 486–87  
UPDATE (RDM Logical view)  
    487–90  
UPDATE (TOTAL File view) 491–  
    92  
UPPERCASE 131, 493–94  
    summary 84  
uppercase restriction  
    CSIOPTNS 165  
uppercase translation  
    hierarchy 131  
    systemwide hierarchy 60  
USAGE 495–96  
    summary 85  
USER 497  
    summary 85  
user name See USER  
user word table  
    size limit 536  
USERWORDS 498  
    summary 85

## V

valid number 339  
valid numbers See NUMERIC  
Validity Status Indicator 505  
VALUE 499–500  
    summary 85  
value of PI See PI  
variable dimensions See DBCS  
    (Statement)  
variable name  
    limits 135  
variable values See LET  
variables  
    allocating small scalar 416  
    assigning a value 308  
    extending 346  
    limit 416  
    naming 415, 457  
    passed 542  
    removing characters 474  
    updated with External DO 530

- vertical bar ( | ) 23
  - inserting comments 39
- VIDEO 128
- VIEW 501–4
  - summary 85
- Vocabulary Work Area
  - releasing 531
  - size limits 533
- VSAM files
  - control intervals 550
- VSI See Validity Status Indicator
- VSI (Function) 505
  - summary 85

**W**

- WAIT 506–7
  - summary 85
- WHEN-END 508–9
  - summary 85
- WHILE-END 510–11
  - NEXT 335
  - summary 85
- window mode 570, 569–73
  - CONVERSE 569
  - limitations 571
  - program initiated 569
  - terminating 573
  - terminating 573

**Z**

- ZERO 512
  - summary 85
- zero-length text value See NULL

